Position and Communication Papers of the 16th Conference on Computer Science and Intelligence Systems pp. 55–60
DOI: 10.15439/2021F111
ISSN 2300-5963 ACSIS, Vol. 26

Towards Evolutionary Emergence

Jörg Bremer
Department of Computing Science
Carl von Ossietzky University
Oldenburg, Germany
joerg.bremer@uni-oldenburg.de

Sebastian Lehnhoff
Department of Computing Science
Carl von Ossietzky University
Oldenburg, Germany
sebastian.lehnhoff@uni-oldenburg.de

***Abstract*—With the upcoming era of large-scale, complex cyber-physical systems, also the demand for decentralized and self-organizing algorithms for coordination rises. Often such algorithms rely on emergent behavior; local observations and decisions aggregate to some global behavior without any apparent, explicitly programmed rule. Systematically designing these algorithms targeted for a new orchestration or optimization task is, at best, tedious and error prone. Suitable and widely applicable design patterns are scarce so far. We opt for a machine learning based approach that learns the necessary mechanisms for targeted emergent behavior automatically. To achieve this, we use Cartesian genetic programming. As an example that demonstrates the general applicability of this idea, we trained a swarm-based optimization heuristics and present first results showing that the learned swarm behavior is significantly better than just random search. We also discuss the encountered pitfalls and remaining challenges on the research agenda.**

I. Introduction

CYBER-PHYSICAL systems (CPS) are equipped with a steadily increasing degree of autonomy (cf. [1], [2]). The technical viability of such systems has already achieved broad attention (see for example [3]). Often, the autonomy in CPS emerges from self-organization principles that are used for coordination as well as from integrating artificial intelligence (AI) -enabled algorithms – as also stipulated in [4] for the example of the European Union. Today's cyber-physical systems already comprise a huge number of physical operation and sensing equipment that has to be orchestrated for secure and reliable operation – prominent examples are the electric energy grid, modern transportation systems, or environmental management systems [5].

As yet, often human operators monitor and control a hierarchically organized CPS and aggregate information from lower level subsystems. Supervisory control and data acquisition (SCADA) systems – as an example from the energy sector – provide a view on and allow for control of a decentralized process and are thus a state-of-the-art means [6].

As complexity grows, more autonomy is desirable in future CPS [7]; desiring for algorithms with self-*-properties [8]. A targeted design of algorithms with specific emergent behavior is difficult to achieve, especially with standard programming languages [9]. Design patterns like [10], [11] may ease the design process, but are often limited in applicability. There are meta-heuristics like the combinatorial optimization heuristic for decentralized agents [12] that are best on self-organization principles, but they need to be manually adapted to each new use case. Having a systematic methodology describing at the design time the construction of self-organizing algorithms or systems step-by-step, would be highly desirable but is hard to achieve for general applicability [13]. Few paradigms and pattern on a rather high abstraction level exist, but the individual adaption to a specific algorithmic or functional goals is left to the designer [13]. Nevertheless, as soon as changes in the system occur, manual adaption might be necessary. The concept of controlled self-organization addresses this issue by introducing a observer-controller architecture for automated correction of self-organized behavior of the system. The concept works well, if correction can be achieved by (re-)parametrizing the self-organizing entities/ agents according to changes in the environment. Sometimes, it might be necessary to change the algorithmic behavior on a level that needs a redesign.

For the initial design of an algorithm addressing a given task by self-organizing mechanism as well as for automated redesign at runtime for proper situational tracking, we propose an automated design of emergent behavior by machine learning approaches. As this goal is a huge field with many aspect to be addressed, we here start by discussing a first example: the applicability of Cartesian genetic programming [14] to the automated design of a swarm-based optimization algorithm for solving global optimization problems.

Thus, we propose to choose a different approach for designing purposeful emergence in self-organizing systems by using machine learning. Machine learning in multi-agent systems is already used for problems that are difficult to solve with preprogrammed agent behavior. The agents must instead discover a solution to the problem on their own, using machine learning [15]; often by reinforcement learning. We go for automatically discovering mechanisms for emergent behavior and self-organization by genetic programming [16]. In this way, we learn control programs for individually acting entities in a decentralized system with the goal to jointly solve a specific problem. As test scenario, we started with swarm-based optimization.

The rest of the paper is organized as follows. We start with a brief review on related work with a focus on multi-agent reinforcement learning and Cartesian genetic programming that we use for learning control programs for particles in our optimization swarm. After describing our share of pitfalls on the way to the first successfully trained swarms, we present

preliminary results comparing our swarm with random search and real particle swarm optimization.

II. Related Work

In general, machine learning algorithms automatically build a mathematical model using sample data. These models are then used to make decisions without a need for specifically programming rules to make these decisions. Starting from the first works of [17] many different algorithms and approaches have been developed. Among them are reinforcement learning [18], [19], classifiers like support vector machines [20], or artificial neural networks [21], to name just a few. A good overview can for example be found in [22].

Reinforcement learning is often applied in intelligent agents for learning to take appropriate actions based on observations from the environment that the agent interacts with [18]. An extension is multi-agent reinforcement learning (MARL) [23]. In MARL, many agents independently learn how to decide on the most rewarding action in a dynamic environment that is disturbed by the other agents. Many MARL algorithms are designed for static and thus stateless games [15]. But, also use cases for cooperative games are scrutinized and may generate emergent behavior [24], [25]. Nevertheless, the application is limited as agents still just learn to choose from a predefined set of (singular) actions [23].

A subset of machine learning algorithms is made up by a special type of evolutionary algorithms. Genetic programming (GP) is used to discover solutions to problems automatically by using evolutionary mechanisms like random mutation, crossover, a fitness function, and multiple generations of evolution. Alan Turing was probably the first to raise the question, whether programs might be evolved by something like evolution [26]. After a first implementation by [27] for logical functions represented as tree programs, many improvements were made [28]–[31].

One of this improvements is the use of a special phenotype representation that allows leaving computational nodes unused. In general, Cartesian genetic programming (CGP) is a more efficient version of genetic programming and encodes computer programs as graph representation [32]. CGP is an enhancement of a method originally developed for evolving digital circuits [33], [34]. CGP already demonstrated its capabilities in synthesizing complex functions in several different use cases for example for image processing [35], neural network training [36], or for the synthesis of Bent functions for cryptography [37].

III. Learning emergence with Cartesian genetic programming

Our goal was to automatically generate a swarm-based heuristics for optimization similar to the particle swarm algorithm, i. e. to derive a swarm of individually acting particles that may include observations from neighboring particles into their own move decisions. To achieve this, we implemented particles that can be equipped with a control program learned by CGP. Figure 1 shows the general architecture. A swarm

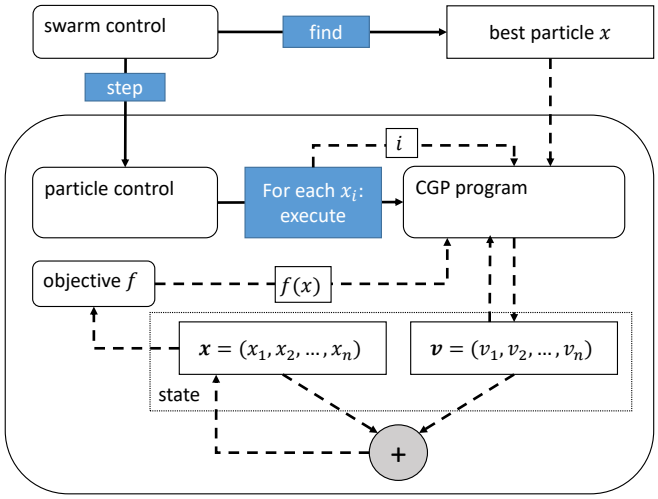

Fig. 1. General architecture of the swarm and the incorporated particles with the embedded CGP program.

consists of an arbitrary number of particles. In each iteration during optimization, each particle is stepped by a global swarm control; just like PSO. The global control is also responsible for ranking the particles and detecting the best one (in terms of fitness). When a particle is stepped, the CGP program that determines the new position of the particle is executed and the new fitness value is calculated. We experimented with different input to the CGP program and different internal particle states as described later

Currently we are only considering the observation of other particles in the swarm. Two succeeding stages of extension will be the integration of inter-entity coordination (1) by using stigmergy and (2) by communication by exchanging messages. Finally, we are opting for problem solving with multi-agent systems.

When learning the control program by CGP, the same swarm setting is used. For each CGP solution candidate, several swarms were set up. Each particle was equipped with the solution candidate program. Each swarm was run for several iterations. Finally, the mean achieved optimization result evaluated the solution candidate.

Cartesian genetic programming is an advanced form of genetic programming (GP) designed to evolve acyclic graphs [38]. The nodes are indexed by their Cartesian coordinates and represent functions of a computational structure (the graph) [39]. Many traditional GP approaches suffered from the so called bloat effect [40] – programs steadily growing in complexity without any significant objective improvement [41]. CGP does not suffer from this problem [40].

A chromosome comprising function as well as connection genes and output genes encodes the computational graph that represents the executable program. Figure 2 shows an example with six computational nodes, two inputs and two outputs. The gene of a function represents the index in an associated lookup-table (0 to 3 in the example). Each computation node is encoded by a gene sequence consisting of the function

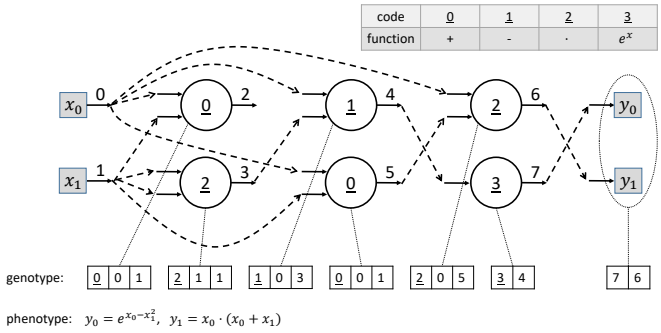

Fig. 2. Computational graph and its genotype representation in Cartesian genetic programming.

look-up index and the connected input (or output of another computation node) that is fed into the function. Thus, the length of each function node gene sequence is $n + 1$ with n being the arity of the function. The graph in traditional CGP is acyclic. Parameters that are fed into a computation node may only be collected from previous nodes or from the inputs into the system. Outputs are connected to any computation node output or directly to any input. Not all outputs of computational nodes are used as input for other functions. In fact, usually many of such unused computational nodes occur in evolved CGP [33]. These nodes are inactive, do not contribute to the encoded program's output, and are not executed during interpretation of the program. In this way, phenotypes are of variable length whereas the size of the chromosome is static.

A computational graph in CGP is typically evolved using a $(1 + \lambda)$-evolution strategy, i. e. with probabilistic mutation but no crossover [42]. CGP allows for unused nodes. Thus, the maximum number of nodes is a priori specified. It has been shown to be advantageous to evolution to overestimate the number of nodes due to an induced higher genetic drift [39].

For our experiments, we used an extension to the ECJ-toolkit [43], [44]. In addition to the traditional integer representation as in Fig. 2, the ECJ version also supports a real-valued representation. For each gene, alleles are allowed to range from $[0, 1]$. Prior to executing the program, all real values are rounded back to integer for interpretation as described above. With real-valued encoding, it becomes possible to apply a real-valued crossover operator. In this way, the performance of convergence is significantly improved at least for regression and [45]. In the integer-encoded case, crossover is usually left out. Nevertheless, more operators are possible with continuous encoding and discovering improved genetic operators for other problems remains an open area of research. We chose to use real-valued encoding.

For learning the internal particle control, we started by setting up a standard CGP scenario. For a start, as function set we chose the four basic arithmetic operations, a generator for normal distributed random numbers, the classical if-then-else-statement, and the set of standard order relations. As input, we gave the current position (in search space), the current objective value, and the position of the best particle. The output of the program was set to be the new particle position. Initially, we introduced an additional parameter $\boldsymbol{v}$ meant to be comparable to the velocity in particle swarm optimization [46], that was output and input to the next iteration as well. In this way, it was meant to enable the particle have a more complex inner state apart from the mere position. But, we were not able to make train CPG to make any targeted use of it. Thus, we changed it to be an automatic increment of the current position.

Instead, we extended the functions set by a function that is able to determine the current rank of the particle (compared with all other). Moreover, the numbers 0-9 were given as constant functions. In many training process we observed that CGP learned to construct needed constants by itself. This was for example achieved by using the if-statement to construct a 1 and then adding it up several times. Usually, this seemed to be a waste of necessary evolutions as well as of needed computation nodes. With introducing the constants, CGP could use the numbers directly. A further improvement was to reuse the same learned program for all dimensions of a multi-variate problem. Figure 1 shows the final architecture of a particle and its embedding in the swarm.

The next challenge was the decision for the objective function. First, we tried evaluating the fitness of a swarm by a single optimization problem. The swarm solves each problem several times and the mean achieved residual problem error is taken to evaluate the performance of the swarm in solving the problem. This approach failed, because CGP learned to solve the given optimization problem directly and made the swarm output the problem solution hard-coded. Actually, this was to be expected. With the next try, we handed a bunch of different optimization problems with optimal solutions at different positions – otherwise it would have resulted in a directly learned result again. With a given set of objectives that are all to be solved independently by the swarm, a sort of swarm behavior could already be generated – but not as expected. The swarm learned to move along a trace that passes through all the optima of the different problems. Again, this was not optimization. In order to tackle this problem, we introduced a random offset. For each problem instance f_i, a random offset $\boldsymbol{r}$ uniformly sampled from the problem domain was generated and added to $\boldsymbol{x}$. The offset is fixed for one training episode. So the swarm solves $f_i(\boldsymbol{x} + \boldsymbol{r})$ resulting in a randomly translated optimum $\boldsymbol{x}^*$. Now we were able to observe an optimization behavior within the trained swarms.

When just using the goodness of the optimization results as criterion for training, the achieved swarm behavior resembles more or less a random search. As our goal was to generate a swarm behavior that exhibits some emergent characteristics and shows self-organization, further criteria evaluating these characteristics need to be added.

Criteria for quantifying emergence are for example known from biology [47] or neuroscience [48]. Applications in computing science are scarce. An example for detecting emergence

in technical systems can be found in [49]. Many fractal analysis tools from chaos theory have a rather high computational complexity. At least, for the application within an objective function for training that has to be called millions of times. For our experiments we tested the so called correlation length as known from fitness landscape analysis [50]. When analyzing fitness landscapes, the correlation length is a criterion that measures the number of iterations after which the majority of succeeding solutions is statistically no longer correlated. It is calculated by $\lambda = -\frac{1}{\ln(\rho(1))}$ from the autocorrelation

$$\rho(\sigma) = \frac{E[f_k f_{k+\sigma}] - E[f_k]E[f_{k+\sigma}]}{V[f_k]} \tag{1}$$

of a series of consecutively sampled objective values f_{k_m}. When using the inverse version, we can maximize this distance. As additional indicators for desired swarm behavior we used the improvement relation

$$r_{\text{imp}} = \frac{1 + \frac{n_{\text{dec}} - n_{\text{inc}}}{n_{\text{dec}} + n_{\text{inc}}}}{2} \tag{2}$$

to maximize the number of improvements n_{inc} over decreasing optimization steps n_{dec}. Finally, we integrated the eventually reached swarm diameter to measure contraction. All criteria were combined in a scalarization approach.

IV. Results

For our experiments we used an islanding model for CGP training with two $(\mu + \lambda)$-ES. One was set to $\mu = 20$ and $\lambda = 100$ with a mutation probability of 0.04. The other was set to $\mu = 8$ and $\lambda = 16$ with a mutation probability of 0.4. Thus, we had a rather steadily evolving ES sending individuals every 1000 iteration and a small rather fast-paced, fluctuating one sending every 100 iterations; thus ensuring liveliness in exploration. The number of nodes was set to 20. As training optimization problems we used Rosenbrock, Bohachevsky, Alpine and Booth [51]. Because each candidate has to be evaluated several times for each of these functions, we limited the number of swarm iterations during the learning phase to 200. The number of particles was set to 5 during training due to performance issues.

Table I and II show the best result. We compared the learned optimization algorithm with a random search and with a real PSO. Random search was our bottom line that needs to be beaten. Table I compares the performance of the swarm, achieved with the number of particles set to 10 and with a budget of 10000 objective evaluations. The performance was tested on six different objective functions; three of which had not been used for learning. Compared with the pure random search, the learned optimization algorithm already behaves rater good, except for the Booth function. For the Rosenbrock function (4-dimensional) and the Six Hump Camel Back functions (2-dimensional), it is already competitive to the PSO. Table II shows the results when using a budget of 200000 objective evaluations; demonstrating that the learned algorithm is significantly better than random search.

In order to detect emergent behavior or at least to distinguish from pure random behavior in the system, we did a quick analysis using two criteria: The correlation dimension [52] and the Hurst exponent [53], [54]. The correlation dimension is a characteristic measure describing the geometry of chaotic attractors. One of the main applications of the Grassberger-Procaccia-algorithm is to distinguish between stochastic and deterministically chaotic time sequences [55]. We use it to analyze the fitness sequence generated along the path of particles. Table III shows example results for some test runs revealing that the behavior of the particles in the learned algorithm behave similar to the ones from PSO when attracted from good solutions. Each run reflects a different objective function. Although, when attacking function 4 from De Jong's test suite [56] which incorporates noise, the learned particle behavior seems to be attracted from more local optima at the same time (larger correlation dimension). The random approach shows no attraction behavior at all.

The Hurst exponent is a measure for the long-term memory of a time series. In this way the long-term statistical dependencies (excluding dependencies from cycles) seen in the series are evaluated [57]. A Hurst exponent of 0.5 denotes white noise. Larger values denote positive dependency, smaller negative dependency. The results in Table IV suggest that the PSO as well as the learned algorithm show a behavior of systematically improving solutions whereas the random search (as expected) exhibits mostly white noise.

V. Conclusion and further work

With the upcoming era of large scale cyber physical systems, the need for controlling numerous entities will in future most likely be accompanied by a growing demand of self-organizing algorithms. We presented a first approach to learn emergent swarm behavior. In a first step, individuals of a swarm were trained to jointly solve global optimization on arbitrary problem instances. So far, mere observation of other swarm members was incorporated. Nevertheless, CGP already was able to come up with solutions that are probable better than a mere random search.

Recently, recurrent CGP has been developed to foster the evolution of recurrent artificial neural networks [58]. For some other use cases, the recurrent version also showed superior performance [42]. On the other hand, necessary control and reduction of the number of recurrent connections introduces new challenges into learning [42]. Nevertheless, one of the next tasks will be to test this version for our use case. Other variants also provide promising extensions or modification [59].

Looking at the mid-term agenda, several challenges still have to be addressed.

- The question for detecting the desired emergent behavior is still open to future research.
- Moreover, if the desired emergent behavior is present, it needs to be quantified to generate appropriate guidance for sampling new solutions.
- What is the best objective function? Obviously, it is a mix of different criteria that would best be addressed

TABLE I
COMPARISON OF THE BEST LEARNED ALGORITHM WITH RANDOM SEARCH AND PSO WHEN USING A BUDGET OF 10.000 OBJECTIVE EVALUATIONS.

function	learned algorithm	random	PSO
Sphere	$5.555 \times 10^{-3} \pm 2.865 \times 10^{-3}$	$1.638 \times 10^{-2} \pm 1.667 \times 10^{-2}$	$4.692 \times 10^{-5} \pm 1.251 \times 10^{-4}$
Rosenbrock	$2.928 \times 10^{-1} \pm 1.824 \times 10^{-1}$	$1.06 \times 10^{-1} \pm 1.187 \times 10^{-1}$	$2.351 \times 10^{-1} \pm 1.045 \times 10^{0}$
Alpine	$1.822 \times 10^{-1} \pm 3.984 \times 10^{-1}$	$7.27 \times 10^{-3} \pm 6.257 \times 10^{-3}$	$2.335 \times 10^{-4} \pm 4.163 \times 10^{-4}$
Six Hump Camel Back	$-1.013 \times 10^{0} \pm 1.242 \times 10^{-2}$	$-9.927 \times 10^{-1} \pm 4.269 \times 10^{-2}$	$-1.031 \times 10^{0} \pm 9.789 \times 10^{-4}$
Booth	$3.837 \times 10^{0} \pm 5.941 \times 10^{0}$	$2.714 \times 10^{-2} \pm 2.603 \times 10^{-2}$	$3.529 \times 10^{-4} \pm 1.4 \times 10^{-3}$
DeJong f4	$3.478 \times 10^{-5} \pm 5.81 \times 10^{-5}$	$4.114 \times 10^{-4} \pm 8.626 \times 10^{-4}$	$1.365 \times 10^{-9} \pm 3.471 \times 10^{-9}$

TABLE II
COMPARISON OF THE BEST LEARNED ALGORITHM WITH RANDOM SEARCH AND PSO WHEN USING A BUDGET OF 200.000 OBJECTIVE EVALUATIONS.

function	learned algorithm	random	PSO
Sphere	$4.971 \times 10^{-7} \pm 3.86 \times 10^{-7}$	$7.158 \times 10^{-4} \pm 7.738 \times 10^{-4}$	$1.2 \times 10^{-12} \pm 4.33 \times 10^{-12}$
Rosenbrock	$1.668 \times 10^{-5} \pm 1.902 \times 10^{-5}$	$6.514 \times 10^{-3} \pm 6.835 \times 10^{-3}$	$5.476 \times 10^{-10} \pm 1.347 \times 10^{-9}$
Alpine	$7.914 \times 10^{-4} \pm 1.199 \times 10^{-3}$	$1.329 \times 10^{-3} \pm 6.917 \times 10^{-4}$	$6.947 \times 10^{-8} \pm 1.532 \times 10^{-7}$
Six Hump Camel Back	$-1.032 \times 10^{0} \pm 1.798 \times 10^{-6}$	$-1.03 \times 10^{0} \pm 1.453 \times 10^{-3}$	$-1.032 \times 10^{0} \pm 2.323 \times 10^{-12}$
Booth	$1.553 \times 10^{-6} \pm 1.293 \times 10^{-6}$	$1.042 \times 10^{-3} \pm 1.079 \times 10^{-3}$	$1.127 \times 10^{-11} \pm 3.635 \times 10^{-11}$
DeJong f4	$3.85 \times 10^{-13} \pm 6.305 \times 10^{-13}$	$4.378 \times 10^{-7} \pm 8.11 \times 10^{-7}$	$5.304 \times 10^{-20} \pm 2.652 \times 10^{-19}$

TABLE III
FRACTAL CORRELATION DIMENSION AS CRITERION TO DISTINGUISH STOCHASTIC AND DETERMINISTIC BEHAVIOR.

function	learned algorithm	random	PSO
Sphere	2.805	1.135×10^{-15}	2.659
Alpine	0.081	-6.107×10^{-16}	1.447
DeJong f4	2.295	2.928×10^{-16}	0.276

TABLE IV
HURST EXPONENT AS INDICATOR FOR LONG TERM MEMORY OF THE SWARM'S DYNAMIC SYSTEM.

function	learned algorithm	random	PSO
Sphere	0.936	0.556	0.872
Alpine	0.933	0.544	0.901
DeJong f4	0.949	0.497	0.758

in a multi-objective approach. In general, CGP could be solved as multi-objective optimization problem, but this would most likely generate severe performance problems.

- One major performance issue is the objective function. For each evaluation of a CGP solution candidate, an optimization procedure has to be run several times and different evaluation criteria have to be calculated.
- Finally, the question for the best set of functions is still open. Presumably, this set can be divided into an always necessary base set and problem specific extensions.

Then, further steps will be the inclusion of first stigmergy and second message-based information exchange. First simple tests of evolving agent-based negotiations via message are already promising for two agents.

REFERENCES

[1] M. Jipp and P. L. Ackerman, "The impact of higher levels of automation on performance and situation awareness," *Journal of Cognitive Engineering and Decision Making*, vol. 10, no. 2, pp. 138–166, 2016.

[2] T. B. Sheridan and R. Parasuraman, "Human-automation interaction," *Reviews of Human Factors and Ergonomics*, vol. 1, no. 1, pp. 89–129, 2016.

[3] R. Parasuraman and C. D. Wickens, "Humans: still vital after all these years of automation," *Human factors*, vol. 50, no. 3, pp. 511–520, 2008.

[4] European Commission, " Draft Ethics Guidelines for Trustworthy AI ," European Commission, Tech. Rep., 12 2018.

[5] B. Rapp, A. Solsbach, T. Mahmoud, A. Memari, and J. Bremer, "It-for-green: Next generation cemis for environmental, energy and resource management," in *EnviroInfo 2011 - Innovations in Sharing Environmental Observation and Information, Proceedings of the 25th EnviroInfo Conference 'Environmental Informatics'*, W. Pillmann, S. Schade, and P. Smits, Eds. Shaker Verlag, 2011, pp. 573 – 581.

[6] L. Cardwell and A. Shebanow, "The efficacy and challenges of scada and smart grid integration," *Journal of Cyber Security and Information Systems*, vol. 1, no. 3, pp. 1–7, 2016.

[7] D. W. McKee, S. J. Clement, J. Almutairi, and J. Xu, "Survey of advances and challenges in intelligent autonomy for distributed cyber-physical systems," *CAAI Transactions on Intelligence Technology*, vol. 3, no. 2, pp. 75–82, 2018.

[8] J. Collier, "Fundamental properties of self-organization," *Causality, emergence, self-organisation*, pp. 287–302, 2003.

[9] H. Parzyjegla, A. Schröter, E. Seib, S. Holzapfel, M. Wander, J. Richling, A. Wacker, H.-U. Heiß, G. Mühl, and T. Weis, "Model-driven development of self-organising control applications," in *Organic Computing – A Paradigm Shift for Complex Systems*. Springer, 2011, pp. 131–144.

[10] J. Dormans *et al.*, *Engineering emergence: applied theory for game design*. Universiteit van Amsterdam [Host], 2012.

[11] M. Parhizkar, G. D. M. Serugendo, and S. Hassas, "Leaders and followers: a design pattern for second-order emergence," in *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2019, pp. 269–270.

[12] C. Hinrichs and M. Sonnenschein, "A distributed combinatorial optimisation heuristic for the scheduling of energy resources represented by self-interested agents," *International Journal of Bio-Inspired Computation*, vol. 10, no. 2, pp. 69–78, 2017.

[13] C. Prehofer and C. Bettstetter, "Self-organization in communication networks: principles and design paradigms," *IEEE Communications Magazine*, vol. 43, no. 7, pp. 78–85, 2005.

[14] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, 2008, pp. 2701–2726.

[15] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010.

[16] J. R. Koza and R. Poli, "Genetic programming," in *Search methodologies*. Springer, 2005, pp. 127–164.

[17] D. O. Hebb, "The organization of behavior; a neuropsycholocigal theory," *A Wiley Book in Clinical Psychology*, vol. 62, p. 78, 1949.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[20] D. A. Pisner and D. M. Schnyer, "Support vector machine," in *Machine Learning*. Elsevier, 2020, pp. 101–121.

[21] M. Van Gerven and S. Bohte, "Artificial neural networks as models of neural information processing," *Frontiers in Computational Neuroscience*, vol. 11, p. 114, 2017.

[22] A. Burkov, *The hundred-page machine learning book*. Andriy Burkov Canada, 2019, vol. 1.

[23] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[24] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[25] F. Martinez-Gil, M. Lozano, and F. Fernandez, "Emergent behaviors and scalability for multi-agent reinforcement learning-based pedestrian models," *Simulation Modelling Practice and Theory*, vol. 74, pp. 117–133, 2017.

[26] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 49, no. 236, pp. 433–460, Oct. 1950.

[27] R. Forsyth, "BEAGLE a Darwinian approach to pattern recognition," *Kybernetes*, vol. 10, no. 3, pp. 159–166, 1981.

[28] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of an International Conference on Genetic Algorithms and the Applications*, J. J. Grefenstette, Ed., Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 Jul. 1985, pp. 183–187.

[29] J. R. Koza, "Non-linear genetic algorithms for solving problems," United States Patent 4935877, 19 Jun. 1990, filed may 20, 1988, issued june 19, 1990, 4,935,877. Australian patent 611,350 issued september 21, 1991. Canadian patent 1,311,561 issued december 15, 1992.

[30] ——, "Hierarchical genetic algorithms operating on populations of computer programs," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, N. S. Sridharan, Ed., vol. 1. Detroit, MI, USA: Morgan Kaufmann, 1989, pp. 768–774.

[31] ——, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[32] L. F. D. P. Sotto, P. Kaufmann, T. Atkinson, R. Kalkreuth, and M. P. Basgalupp, "A study on graph representations for genetic programming," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 931–939. [Online]. Available: https://doi.org/10.1145/3377930.3390234

[33] J. Miller, *Cartesian Genetic Programming*, 06 2003, vol. 43.

[34] J. F. Miller, P. Thomson, and T. Fogarty, "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study," *Genetic algorithms and evolution strategies in engineering and computer science*, pp. 105–131, 1997.

[35] S. Harding, J. Leitner, and J. Schmidhuber, "Cartesian genetic programming for image processing," in *Genetic programming theory and practice X*. Springer, 2013, pp. 31–44.

[36] M. M. Khan, A. M. Ahmad, G. M. Khan, and J. F. Miller, "Fast learning neural networks using cartesian genetic programming," *Neurocomputing*, vol. 121, pp. 274–289, 2013.

[37] R. Hrbacek and V. Dvorak, "Bent function synthesis by means of cartesian genetic programming," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2014, pp. 414–423.

[38] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[39] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[40] J. Miller, "What bloat? cartesian genetic programming on boolean problems," in *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*. San Francisco, California, USA, 2001, pp. 295–302.

[41] A. J. Turner and J. F. Miller, "Cartesian genetic programming: Why no bloat?" in *European Conference on Genetic Programming*. Springer, 2014, pp. 222–233.

[42] ——, "Recurrent cartesian genetic programming applied to famous mathematical sequences," in *Proceedings of the Seventh York Doctoral Symposium on Computer Science & Electronics*, 2014, pp. 37–46.

[43] E. O. Scott and S. Luke, "Ecj at 20: toward a general metaheuristics toolkit," in *Proceedings of the genetic and evolutionary computation conference companion*, 2019, pp. 1391–1398.

[44] J. Miller and N. C. Series, "Resources for cartesian genetic programming," *Cartesian Genetic Programming*, p. 337.

[45] J. Clegg, J. A. Walker, and J. F. Miller, "A new crossover technique for cartesian genetic programming," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, pp. 1580–1587.

[46] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[47] V. Balaban, S. Lim, G. Gupta, J. Boedicker, and P. Bogdan, "Quantifying emergence and self-organisation of enterobacter cloacae microbial communities," *Scientific reports*, vol. 8, no. 1, pp. 1–9, 2018.

[48] E. P. Hoel, L. Albantakis, and G. Tononi, "Quantifying causal emergence shows that macro can beat micro," *Proceedings of the National Academy of Sciences*, vol. 110, no. 49, pp. 19 790–19 795, 2013. [Online]. Available: https://www.pnas.org/content/110/49/19790

[49] A. Shahbakhsh and A. Nieße, "Modeling multimodal energy systems," *Automatisierungstechnik : AT*, vol. 67, no. 11, pp. 893–903, 2019.

[50] J.-P. Watson, "An introduction to fitness landscape analysis and cost models for local search," in *Handbook of metaheuristics*. Springer, 2010, pp. 599–623.

[51] M. Jamil, X.-S. Yang, and H.-J. Zepernick, "8 - test functions for global optimization: A comprehensive survey," in *Swarm Intelligence and Bio-Inspired Computation*, X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, and M. Karamanoglu, Eds. Oxford: Elsevier, 2013, pp. 193–222.

[52] P. Grassberger and I. Procaccia, "Characterization of strange attractors," *Physical review letters*, vol. 50, no. 5, p. 346, 1983.

[53] H. E. Hurst, "The problem of long-term storage in reservoirs," *Hydrological Sciences Journal*, vol. 1, no. 3, pp. 13–27, 1956.

[54] ——, "A suggested statistical model of some time series which occur in nature," *Nature*, vol. 180, no. 4584, pp. 494–494, 1957.

[55] P. Grassberger, T. Schreiber, and C. Schaffrath, "Nonlinear time sequence analysis," *International Journal of Bifurcation and Chaos*, vol. 1, no. 03, pp. 521–547, 1991.

[56] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, Ann Arbor, 1975.

[57] R. Weron, "Estimating long-range dependence: finite sample properties and confidence intervals," *Physica A: Statistical Mechanics and its Applications*, vol. 312, no. 1-2, pp. 285–299, 2002.

[58] A. J. Turner and J. F. Miller, "Recurrent cartesian genetic programming of artificial neural networks," *Genetic Programming and Evolvable Machines*, vol. 18, no. 2, pp. 185–212, 2017.

[59] A. Manazir and K. Raza, "Recent developments in cartesian genetic programming and its variants," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.