Auton Agent Multi-Agent Syst (2018) 32:275–311
https://doi.org/10.1007/s10458-017-9380-x

Clonal plasticity: an autonomic mechanism for multi-agent systems to self-diversify

Vivek Nallur[1] · **Siobhán Clarke**[1]

Published online: 7 December 2017

Abstract Diversity has long been used as a design tactic in computer systems to achieve various properties. Multi-agent systems, in particular, have utilized diversity to achieve aggregate properties such as efficiency of resource allocations, and fairness in these allocations. However, diversity has usually been introduced manually by the system designer. This paper proposes a decentralized technique, clonal plasticity, that makes homogeneous agents self-diversify, in an autonomic way. We show that clonal plasticity is competitive with manual diversification, at achieving efficient resource allocations and fairness.

1 Introduction

Socio-technical systems are all around us, and increasingly computers are an intrinsic part of these systems [1]. Smart-grids [2], vehicular ad-hoc networks [3,4], smart buildings [5], e-procurement [6], cloud computing [7], healthcare [8], and transport [9] are some examples of domains where humans interact with computer systems to achieve a particular goal. Due to the speed, complexity, and frequency of decision-making involved in these domains, computers are often required to be autonomic and adaptive to the dynamic environment around them. Indeed this is a foundational reason why we create self-adaptive systems. Socio-technical systems were first defined by Emery and Trist [10] to be a complex interaction between humans, machines and their joint environment. This means that design choices made for the construction/selection of computer programs would also significantly affect the humans in the system.

A commonly accepted definition of an agent is given by Woolridge [11]–"An agent is a computer system that is situated in some environment, and that is capable of autonomous

✉ Vivek Nallur
vivek.nallur@ucd.ie

Siobhán Clarke
siobhan.clarke@scss.tcd.ie

[1] School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland

action in this environment in order to meet its design objectives". Computer systems that function in socio-technical systems are increasingly agents that are, by definition, autonomous. However, in a socio-technical system filled with such agents, each with its own set of constraints, goals and strategies, the resulting collective system may not be predictable. Rather, the multiplicity of agents, their possible actions, and corresponding feedback loops could result in a complex system [12]. This makes the aggregate outcome of individual actions dynamic and difficult to predict. An action taken by an agent at a particular point in time may have an outcome that is vastly different from the same action at a different point in time. In many such systems, agents make (or recommend) decisions that result in allocation of *common pool* resources [13]. That is, resources in a social context are divided amongst actors/entities in a manner that achieves some objective. In such systems, less-than-ideal outcomes are most clearly visible where some resource is being allocated, and there is no centralized mechanism to ensure that autonomous actions by each agent does not result in a disastrous allocation, at the aggregate level. This is specially so in cases where the agents are not, *apriori*, designed to be cooperative (*e.g.,* traffic jams, where each vehicle is exhibiting individually rational behaviour). In many domains, the natural model is a competitive set of agents where each agent attempts to fulfill its goals, without regard to the effect it has on the system-as-a-whole. Indeed, if the system is large enough, the agent cannot even view the system-as-a-whole. Examples of such domains include traffic, energy markets, financial markets, etc. In such domains, multi-agent system designers need a mechanism that can harness the multiplicity of agents to produce good outcomes, even where the agents are not explicitly designed to cooperate.

In other words, resource allocation mechanisms in multi-agent systems must guard against some agents grabbing more than their fair share of resources without actually trampling on the autonomy of the agents themselves. Typically, resource allocation in multi-agent systems are analyzed for two types of aggregate outcomes: *(a)* efficiency, and *(b)* social welfare. The issue of efficiency is quantified as *Pareto-Efficiency*: where the total allocation is such that it is not possible to improve the allocation for some agents without making it worse for some other agents. A pareto-efficient allocation, therefore, represents a situation where the solution cannot be bettered, without hurting at least one other agent. Social welfare is usually codified as *envy-freeness*: no agent would rather hold the resource bundle held by another agent [14]. For example, consider a route-choice problem through a city *i.e.*, pick road A or road B to reach a destination. This is a familiar situation in many cities, where multiple humans, utilizing commonly known (albeit coarse-grained) information about traffic flows, along with their own private heuristics (*e.g.*, road A is generally congested from 7:30-8:30 am), make route choices to reach their destination. These agents may use additional machine intelligence in the form of real-time traffic updates through GPS devices/smartphones, and shortest-path algorithms, while making their choices. The reward for this choice is determined dynamically, based on the aggregate number of agents that make a particular choice without prior coordination, *i.e.*, the agent that chooses the 'road less travelled' would experience lower congestion. There is no way to force an agent to make a particular choice, or to choose a particular strategy. However, regardless of the strategy used to pick a route, it must yield a satisfactory outcome for all agents, *at least some of the time*, else some agents would be envious of others. Envious allocations are not a good outcome in socio-technical systems, in contrast to efficiency-based problem formulations, where the only concern is to optimize the traffic flow through that route. As pointed out by *Chevaleyre et al.*, in some systems, it is impossible to allocate resources (roads, in this case) such that the result is both efficient and envy-free [15]. Furthermore, in certain simultaneous games, even solely envy-freeness is impossible to guarantee [16].

This paper is concerned with ameliorating the negative effect of the impossibility of envy-freeness in such games, by enabling long-term fairness through cumulative allocations. It is in this context that we introduce the idea of fairness. We distinguish fairness from envy-freeness, by observing that a series of envious allocations may, over time, turn out to be cummulatively fair. We invoke Rescher's notion of fairness through distributive justice [17] over the total resources allocated amongst all agents. All agents have a legitimate claim on the road, and distributive justice consists of reconciling these claims in the face of plurality of allocation choices. Rescher says, "... a claim represents what an individual ought *ideally* to be given in the light of the principles of justice ..." [18]. In the case of the route-choice problem, this would be the evenness of the rewards (experience lower congestion) that accumulate to drivers, over multiple days. That is, even though it is impossible to ensure that all users in a road network get the same rewards on any given day, it is **sufficient** to ensure that, over time all users get **approximately** the same cummulative reward. The lack of envy-freeness on any given day, is compensated by the distribution of envious allocations over time and across agents.

The contribution of this paper is to present a technique that allows us to achieve both efficiency and fairness. We borrow diversity as a metaphor from ecology, and model the multi-agent-system as an ecosystem, where each agent is concerned in a selfish way, only with its own survival. We had previously shown [19] that **diversity in an ecosystem of competing agents can drive the collective measure of efficiency and fairness higher**. However, the algorithmic diversity demonstrated was achieved via *deliberate human design*. This is not a scalable mechanism (or indeed, a sustainable mechanism) for long-lived, large multi-agent systems consisting of thousands of agents. In this paper, we present a mechanism (clonal plasticity) that allows agents to self-diversify in an autonomic fashion. We evaluate the diversity achieved via clonal plasticity, and reflect on its scalability.

The rest of this paper is structured as follows: First, we review resource allocation in multi-agent systems and the various ways in which diversity has been used in software systems (Sect. 2). We also review various mechanisms of introducing and sustaining diversity used in other domains of computer science (Sect. 3). Next, we make explicit what we mean by diversity in multi-agent systems, and how the diversity present in a multi-agent system can be quantified (Sect. 4). We consider a multi-agent system, where multiple agents act autonomously to secure resources (Sect. 5) and show that increasing the amount of diversity results in an increased distributive justice amongst the agents (Sect. 6). We then introduce a self-tuning mechanism (clonal plasticity) that measurably increases the diversity amongst the agents (Sect. 7) and formalize the expected fairness from clonal plasticity (Sect. 8). We show experimentally that clonal plasticity does increase diversity, compare its effectiveness against manual diversification, and discuss the scalability of clonal plasticity (Sect. 9). We also contrast it against other phenotypic-diversity generation mechanisms(Sect. 10) as well as non-phenotypic diversity generation mechanisms (Sect. 11). We reflect on the limitations and weaknesses of clonal plasticity (Sect. 12) and finally, we conclude the paper (Sect. 13) with a few thoughts on open questions and potential areas of further research.

2 Related work

We first look at the concept of diversity, as transferred from the ecological sciences, and how it has been implemented in computer systems. Then we review the work on decentralized resource allocation in multi-agent systems.

2.1 Diversity in software systems

In the natural world, diversity has long been considered a contributor to stability of natural ecosystems [20,21]. While not sufficient to guarantee the stability of an ecosystem, it is nevertheless considered critically important, due to the *Insurance Hypothesis* [22–24]. Briefly, the Insurance Hypothesis relies on the observation that "differential response by species to a perturbation or variation in environment causes a dampening of the effect of the perturbation". Simply put, an increase in diversity increases the probability that there will exist some species that is able to respond differently to environmental perturbations. The presence of diversity is therefore hypothesized to be an insurance against an ecosystem collapsing due to perturbations.

2.1.1 Security

Researchers have applied the notion of diversity in order to protect systems from attacks and intrusions. The principal use of diversity in these approaches has been in the form of creating a *moving target* for defense. The idea is to reduce the predictability in the structural layout of programs in memory, thus making it more expensive for attackers to create an attack that can work across a range of systems. Multiple stages in the lifecycle of software development such as, *creation* [25], *compilation* [26–29], *deployment of software* [30–33] have benefited from the introduction of variability.

2.1.2 Search and machine learning

Differential response has also been used to explore different parts of a search space, in order to aid search techniques such as genetic algorithms [34] and particle swarm optimization [35,36]. The presence of diversity has been used to create ensemble learners that learn faster than other learners [37]. There has been work in the area of support vector machines that incorporate the notion of diversity to create more accurate active-learning [38,39]. Other computational intelligence techniques such as Artificial Immune Systems have also recognized diversity [40,41] to be integral to its effectiveness. Almost all techniques in the field of search and computational intelligence now explicitly incorporate mechanisms to generate and maintain diversity.

2.1.3 Software engineering

Apart from resilience in the presence of attackers, diversity as a concept has also been investigated for software product lines [42] in the form of design diversity, management of functional diversity using plugins [43] and survivability [44]. Diversity, as a software engineering phenomenon has also been extensively studied in various surveys [45–47].

All of these surveys and papers have considered "differential response by a species" to be the critical element of diversity. However, to the best of our knowledge, we are not aware of any prior work that proposes a systematic technique for diversification of multi-agent systems (MAS). This is critically important since agents in a MAS are being used to control and coordinate complex pieces of software, such as smart-grids [2], vehicular ad-hoc networks [3,4], smart buildings [5], healthcare [8], and transport [9] systems.

2.2 Resource allocation in multi-agent systems

The allocation of resources within a society of agents such that each individual agent's preferences are met forms a multi-disciplinary research area, with inputs from Computer Science and Economics. There are many approaches to realizing an allocation, and each approach has its own properties with regard to three fundamental axes: **(a)** what kind of resources (divisible / indivisible) are being allocated? **(b)** what is the allocation procedure to be followed? and **(c)** what constitutes a good allocation? Each of these axes can be subdivided further:**(i)** how are resources and preferences represented? **(j)** how do agents negotiate amongst themselves? **(k)** what is the computational complexity of the entire allocation process, and so on. Depending on the domain where multi-agent resource allocation (MARA) occurs, different combinations of these properties will take priority over others. The breadth of domains for which MARA has been used is broad enough (network routing [48], air traffic control [49], manufacturing [50,51], transport and logistics [52], e-procurement [53], and even sharing satellite resources [54]) that these properties become the critical factors on which the choice of allocation approach in a particular software system hinges.

2.2.1 Types of resources

Different types of resources may be in contention for allocation. Some resources are divisible (*e.g.,* network bandwidth [48]) whereas others are indivisible (*e.g.,* slot for landing aircraft [49]). Some resources are shareable (*e.g.,* images from a satellite [54]), while others are non-shareable (*e.g.,* resources in a factory [50]). Some domains may even have multiple types of resources, but may have business constraints that only allow certain *bundles* of resources to be allocated (*e.g.,* multi-unit e-procurement [53])

2.2.2 Allocation approaches

Fundamentally, all allocation procedures can be classified as being *centralized* or *decentralized*. Centralized approaches are mostly in the form of combinatorial auctions [55], which may themselves use multiple strategies for trading (*e.g.,* machine-learning [7], game-theory [56]). Decentralized approaches include negotiation protocols such as *Contract-Net* [57] and its variants (Leveled Commitments [58], Constraint Propagation [59]). Centralized approaches have the advantage of being easy to understand and implement, however they suffer from their inability to scale. For a large system, consisting of hundreds or thousands of agents, scalability is a critical issue. Decentralized approaches, on the other hand, scale well to large numbers of agents, which are typical of large socio-technical systems such as traffic on city roads. There have also been attempts to extract the best of both worlds by performing resource allocation in a hybrid manner, with a mix of centralized and decentralized decision making. [60], [61], [62] are all approaches that utilize particle-swarm optimization as a middle ground, with each individual particle behaving in a decentralized manner, along with neighbourhood rules that result in the creation of small hierarchies (partial centralization).

2.2.3 Social welfare

One of the objectives of MARA is the achievement of allocations that reflect the desired preferences of the individual agents. This introduces an axis, where the aggregate social distribution of resources is as important as the efficiency and ease of allocation. Concepts

from social welfare and social choice theory [63] influence the allocation procedures, and the desired ordering of resource allocations. Bentham's utilitarianism [64] is concerned with maximizing the total utility in the system. Other theories such as Rawls's theory of justice [65] advocates an egalitarian allocation which is measured by the utility gained by the agent that is *worst-off*. Other notions of welfare include *elitist social welfare* (measure the utility of the agent that is best-off) and *maximin ordering* (ranks allocations such that the worst-off agent has the highest possible utility) which can be used to evaluate the allocations generated by various schemes.

2.3 Distributive justice

The notion of distributive justice has been dealt with in multiple works in the field of social choice theory. However, in the field of computer science, we are aware of only one thread of research [66,67], that explicitly attempts to ensure distributive justice in multi-agent systems. We view our work as complementary, as we approach the same research theme, but from two different ends of the spectrum. Pitt et al. develop a formal framework for operationalizing distributive justice rules in self-organizing systems [66]. In this work, we present an empirical approach that measures the amount of distributive justice under certain conditions and present a mechanism to autonomically increase this justice in homogeneous systems.

3 Diversity generation mechanisms in literature

As is evident from Sect. 2, diversity has been well-recognized as a mechanism for positively influencing system properties, in many domains. However, there hasn't been a similar wealth of work in actually generating diversity, in a homogeneous system. In the field of security, diversity through randomization [27–33] has been the most dominant tactic. Most work in automated diversity generation has been in the domain of evolutionary and swarm-based algorithms, where populations (and therefore diversity) are a natural solution concept.

3.1 Evolutionary computing

Search techniques such as Evolutionary Algorithms, Genetic Algorithms, Genetic Programming, etc. all rely on population-based approaches along with fitness functions that aim to mimic the natural process of evolution [68]. In all of these techniques, diversity is a key factor in whether the population is able to converge to a good solution. Note that diversity, *per se*, is not a requirement, rather it is a strong observational recommendation. This is echoed by McPhee and Hopper in [69], "Progress in evolution depends fundamentally on the existence of variation of population. Unfortunately, a key problem in many Evolutionary Computation (EC) systems is the loss of diversity through premature convergence". Diversity, hence, is generated or preserved through various approaches. Cobb and Grefenstette introduced a random immigrants approach [70] where random individuals are inserted into the population in every generation. Introduction of specialized selection operators (such as *incest prevention* [71] and *mating restrictions* [72]) also encourage diversity in the population by disallowing genetically similar individuals to mate. Črepinšek et al. [73] classify different strategies for diversity maintenance and diversity control in a population and observe that most methods simply expect an increase in diversity and hence, a better balance between exploration and exploitation. They however caution that not all diversification is useful. Depending on whether more exploration is required or more exploitation is necessary, the

levels of useful diversity will vary. That is, increasing diversity by itself does not necessarily mean that the system will perform better.

3.2 Swarm-based optimization algorithms

Swarm-based algorithms have been used for several combinatorial and optimization problems. Algorithms in this category view swarms as any restrained collection of interacting agents or individuals [74]. Particle Swarm Optimization (PSO) was introduced by Eberhart and Kennedy [75] as a mechanism for optimization of non-linear functions. In a similar vein, swarm-based algorithms such as Ant Colony Optimization [76] and Bee Colony algorithms [74,77] have been proposed as meta-heuristics for solving combinatorial problems in traffic and transportation [78]. The central problem in all of these optimization algorithms is how to avoid local minima in complex search spaces, and achieve global optima quickly. Diversity is usually achieved in these methods using concepts such as 'critical thresholds'[79], 'electro-static charges'[80], etc. The fundamental idea in these methods is to probabilistically avoid dense neighbourhoods by making particles move away from each other upon triggering a certain event/threshold.

3.3 Machine learning

In the field of Machine Learning, generation of diversity has been explicitly considered in ensemble-based methods. Diversity of neural network architectures has been investigated through the process of speciation [81]. In [81], the problem of combining multiple artificial neural networks is tackled via a Darwinian evolutionary process along with *fitness sharing* as the key ingredient that promotes speciation, and thus the creation of diverse neighbourhoods. Their mechanism specifies a population-wide calculation of fitness, and subsequent adjustment of individual-fitness based on the sharing radius. Fitness sharing penalizes individuals in dense neighbourhoods, while individuals in sparse neighbourhoods are promoted, thus increasing the probability that diverse individuals are carried forward into the next generation.
Concept Drift is an important topic in online learning mechanisms, where classifiers that learn on a particular dataset, are subsequently required to classify using variables that have changed with time. Minku et al. ([82]) had shown that while ensembles with low diversity showed lower test errors, they were unable to cope with a dynamic environment and consequent concept drift. Ensembles with higher diversity were able to cope better, regardless of the type of drift encountered. Subsequently, they proposed a method that used four ensembles with two levels of diversity [37] to successfully handle environments, both without drift as well as with concept drift. The levels of diversity are generated using different values for presenting a training example. These different values (drawn from a poisson distribution) result in different ensembles learning different classifications.

4 Algorithmic diversity

Decision-making and resource allocation mechanisms are all realized by various algorithms, and hence we view algorithmic diversity as an appropriate level of granularity for achieving our goals. We define *Algorithmic Diversity* as: a variability in the output produced by a set of algorithms, when faced with the same input/input-sequence. We do not consider differences

in the internal data structures used (stacks, queues, trees, etc.) or control flow implemented (iteration, recursion, continuations, labelled jumps, etc.) for two reasons:

1. Detecting whether two algorithms are the same even when they are exceedingly simple in their construction is so hard that for practical purposes it is infeasible [83]
2. More philosophically, from the perspective of the environment (or other agents), if two systems produce exactly the same output (behaviour), regardless of the input sequence, then they are identical, regardless of their internal construction.

This definition positions us in domains where there are multiple valid answers and no deterministic ways of calculating the optimal response in advance. Autonomous systems fall squarely into this category since the environment in which they operate is dynamic in nature. Traffic flow management systems are a good example of dynamically changing environments. Each vehicle in traffic has (possibly) different source and destination and different strategies in how it gets there (*e.g.,* fastest route, shortest route, least polluted route, etc.). It is in these scenarios that aggregate outcomes such as efficiency and fairness are important. While autonomous agents may negotiate amongst themselves to reach an efficient allocation of road resources, it is not necessary that this allocation is fair. It is in this context that we hypothesize that diversification of algorithms leads to a fairer distribution of rewards amongst the population, regardless of the actual algorithms in use. This hypothesis was first formulated and tested in [19]. In this work, we extend the results as well as introduce an autonomic mechanism to allow agents to self-diversify. We also compare the effectiveness of the autonomic mechanism against the manual efforts earlier.

We use a self-organizing game called the Minority Game (MG) [84] as our exemplar Multi-Agent System in a traffic setting, primarily because it has been well-studied and well-understood. The Minority Game (MG), introduced by Challet and Zhang, consists of an odd number (N) of agents, playing a game in an iterated fashion.[1] At each timestep, each agent chooses from one of two actions (A or B), and the group that is in the minority, wins. Since N is an odd integer, there is guaranteed to be a minority. Winners receive a reward, while losers receive nothing. After each round, all agents are told which action was in the minority. This feedback loop induces the agents to re-evaluate their action for the next iteration.

5 Minority game

While simple in its conception, MG has been used in many fields like econophysics [85–87], multi-agent resource allocation [88,89], emergence of cooperation [90], and heterogeneity [91,92]. The Minority Game (MG) is intuitively applicable to many domains, such as traffic (the driver that chooses the 'less-travelled' road experiences less congestion), packet traffic in networks, ecologies of foraging animals, etc. MG has been acknowledged to be an example of a complex adaptive system, where the individual behaviour is simple to understand, yet the aggregate dynamics are complex and have served as a proxy in multiple domains such as financial markets [93], distributed decision-making in wireless networks [94], and smart-buildings [95].

[1] Note that the exact number of agents does not impact the results, as long as there is a well-defined minority at every stage.

5.1 Description of the game

There is a set of $N = \{1, ..., n\}$ agents (N = odd), each implementing an algorithm $A = \{a_1, ..., a_n\}$. At every iteration of the game G_i, each agent chooses from a binary set $\{A, B\}$. Thus, at each iteration i of the game (G):

$$G_i(a_j) \rightarrow A|B$$

refers to agent j making a binary choice from A or B. If the subset of agents that chose A is smaller than the subset of agents that chose B, then the agents that chose A are said to be in the minority $(minG_i)$ at iteration i. And vice versa. The agents that are in $minG$ for any particular iteration, earn a reward of one unit. The agents that are not in $minG$ receive no reward. If R_i is the reward earned by an agent for a particular iteration, then the total reward earned by an agent after T iterations is given by:

$$\sum_{i=1}^{T} R_i(a)$$

Clearly, the allocation of rewards is most efficient when the size of $minG = N/2$. While the allocation is least efficient when the size of $minG = 1$ (one agent chooses A, and all the rest choose B).

5.2 Measuring algorithm diversity

In MG, each agent shares the same functional objective, to continue to be on the side of the minority, and thereby accumulate a reward. The difference in each agent is the algorithm used to compute the best action (A or B). Depending on the type of algorithm used, differences in initialization parameters could lead to different actions chosen (*e.g.,* in evolutionary or machine learning algorithms). Thus, any algorithm that exhibits a different output-sequence is categorized as a different variety. The calculation of diversity in the game depends only on the variety and balance in the population of agents (implementing those algorithms). This calculation is most famously done by the *Shannon Index* (Eq. 1). The Shannon Index measures the entropy (or uncertainty) in picking an individual of a particular type, from a dataset [96]. Thus, for instance, if all the agents in a MAS implement exactly the same algorithm, the entropy (or uncertainty) will be zero. The higher the diversity in the population, the higher is the Shannon Index. In Eq. 1, R is the number of types in the population, and p is the proportion of individuals in the population with type i.

$$H' = -\sum_{i=1}^{R} p_i \ln p_i \tag{1}$$

The Shannon Index is a measure of both, the *richness of species* (number of different algorithms) and *evenness of species* (abundance of agents implementing an algorithm). We use the Shannon Index to measure how much algorithms differ, in their key parameters. Thus, if a particular initialization parameter (that has an effect on the output-sequence3) can take multiple values, the Shannon Index will measure how many such differentiated types exist and how many individuals implement that type, as a proportion of the total population.

6 Effect of algorithmic diversity

In this section, we re-cap the important aspects of the results from [19]. Diversification of algorithms was performed on two levels: *(a)* Parameter Diversification, and *(b)* Strategy Diversification.[2] Three algorithms were chosen as the candidate algorithms:

1. *BestPlay* Introduced by the creators of the MinorityGame, this is a simple strategy that nevertheless exhibits interesting dynamics at the aggregate level [84]. In this strategy, all players had access to a bounded collective memory (m) of the game, along with a pool of strategies(s). Collective memory is implemented in the form of a history of which action (+1 or -1) was in the minority, at each timestep. Each player drew s strategies randomly from a strategy space, that indicated which action to play next, based on the history of the game. Each player allocated virtual points amongst all its strategies, based on the continually changing history of the game. Strategies that correctly predicted the correct output, regardless of whether they were used or not, received virtual points. Each player, for the next round, used the top-ranking strategy amongst his pool of strategies. Table 1 shows a sample strategy pool used by a player. The size of each strategy depends on m, the memory of the game, while the total number of strategies in the pool depends on the computational power the agent possesses. Table 1 is therefore taken from a game with a memory of size 3, while the agent has a pool size (k) of 4. Typically, each agent in the entire game has the same pool size (k). While it is intuitively obvious that the greater the pool size, the greater the probability of an agent drawing good strategies, it is impossible for an agent to have a pool size that exhaustively covers every possible strategy, since the strategy space grows hyper-exponentially(2^{2^m}) with increase in m.
2. *Evolutionary* Evolutionary algorithms have been used quite successfully in multiple domains [97,98]. As pointed out in [99], evolutionary algorithms are now being compared to human beings, in their ability to solve hard problems. We have implemented an evolutionary algorithm that uses *BestPlay*'s strategy representation as a genome and performs crossover and mutation on it. The strategy is as follows: At every reproduction cycle, the ten most poorly performing agents get rid of their strategies and adopt a combination of strategies from two parents randomly selected from the ten best performing agents. Effectively, two of the ten best performing agents have reproduced to create an offspring that replaces one of the ten worst agents. The offspring mutates the strategies obtained from its parents, by some small mutation probability. This allows the offspring to get better, as well as explore more of the strategy space. The reproduction cycle refers to the rounds after which evolution occurs. A frequency of 20 cycles means that after every 20 rounds of the game, the worst 10 agents discard their strategies and inherit good strategies from 2 of the top 10 agents. These are then mutated according to the agents' own mutation probability.
3. *Roth-Erev* We implement a Reinforcement-Learning algorithm, called Roth-Erev [100]. Reinforcement-Learning(RL) is a class of machine-learning algorithms where an agent tries to learn the optimal action to take, in a certain environment, based on a succession of *action-reward* pairs. Given a set of actions (a) that an agent can take, the Roth-Erev learning algorithm performs the following steps:

   (a) Initialize initial propensities (q) amongst all actions (N)
   (b) Initialize strength (s) of initial propensities
   (c) Initialize recency (φ) as the 'forgetting' parameter

[2] All code and data for this experiment is available at https://bitbucket.org/viveknallur/aamas2016.git.

Table 1 Sample strategy pool for a player

Strategy	Output
[+1 −1 +1]	−1
[−1 −1 −1]	−1
[−1 −1 +1]	+1
[+1 +1 +1]	−1

(d) Initialize epsilon (ε) as the exploration parameter
(e) Choose action (a_k) at time (t), based on q
(f) Based on reward (r_k) for a_k, update q using the following formula:

$$q_j(t+1) = [1-\varphi]q_j(t) + E_j(\varepsilon, N, k, t)$$

$$E_j(\varepsilon, N, k, t) = \begin{cases} r_k(t)[1-\varepsilon] & \text{if } j = k \\ r_k(t)\frac{\varepsilon}{N-1} & \text{otherwise} \end{cases}$$

(g) Probability of choosing action j at time (t) is given by:

$$P_j(t) = \frac{q_j(t)}{\sum_{n=1}^{N}[q_n(t)]}$$

(h) Repeat from step 3e

6.1 Parameter diversity

Parameter diversity refers to diversity in the initialization parameters of the algorithm being implemented by the agents. That is, for any game, all agents still implement the same code, but some parameter in the algorithm is diversified. For the Evolutionary and Roth-Erev algorithms, even minute differences within the initialization parameters can cause the aggregate average payoff to vary. For each of the three families, the following parameters were diversified within the algorithm:

1. Best Play algorithm: k–which is the pool of strategies, a player has (strategies-per-agent).
2. Evolutionary algorithm: The mutation probability that allows an agent to explore the strategy space, by mutating known good strategies.
3. Roth-Erev algorithm: The parameters of *recency* (φ), and *exploration* (ε)

6.2 Strategy diversity

We mix agents implementing different algorithms (strategies) into one population. The diversification here takes place on two levels, parameter-diversification (as before), as well as proportion of population implementing a particular strategy. The total population of agents in all scenarios is kept constant, but the proportion of agents playing a particular strategy is varied. This leads to different levels of diversification, for each combination of strategies. The parameters that were varied were as follows:

1. *Number of algorithms* This reflects the effective number of families (or *genera*) of species that are present in the population.
2. *Proportion of population implementing an algorithm* Depending on the relative number of each specie, the aggregate reward in the game changes.

Table 2 Experimental constants for minority game

Variable	Value
Population size	501
Simulation period (rounds)	2000
Repetition of variation	100

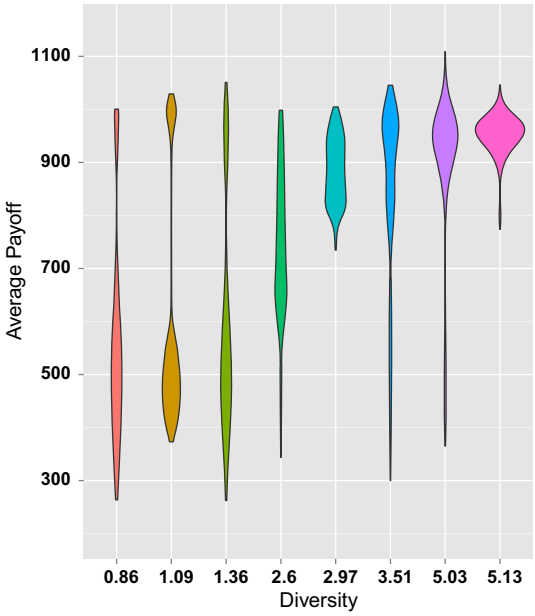

Fig. 1 Payoff across all diversification

3. *Parameters within algorithms that were diversified* Variation in critical parameters for an algorithm results in effectively creating a new specie, since the functional pathway (and resultant output) begins to change depending on the environment.

Table 2 shows the constant factors in each experiment. Each minority game was played with a population size of 501 (N) agents, through a simulation time period of 2000 steps (T). For each variation in the experimental setup, the data is reported as an average of 100 simulations.

6.3 Results

We combine the results from all of the diversification to illustrate the comparative rewards achieved at different levels of diversity. The graphs show the distribution of the population achieving a particular level of average reward. The 'fatter' the graph, the more evenly rewards are distributed, and vice-versa. Figure 1 clearly shows that the higher the diversity level, the fairer the average reward. Not only is the reward fairer, it is also higher, which is an important consideration for resource allocation strategies. Since it is the same three algorithms that are mixed in different proportions to achieve different levels of diversity, it is clear that diversity-level is the distinguishing factor in the level of rewards.

As was pointed out earlier in this section, MG represents a class of games where envy-freeness is impossible to achieve. After any given round, a majority of the agents are envious

of the allocations secured by the minority. Algorithmic Diversification offers an alternate resolution, in such games, where multiple iterations of the game result in approximate envy-freeness. That is, on multiple iterations of an envious game, the introduction of algorithmic diversity, moves all the agents towards receiving approximately the same amount of reward As seen in Fig. 1, the increase in diversity results in increased fairness, which can be seen as a proxy for envy-freeness. This occurs due to a 'rotation' of agents that receive a reward, through the multiple iterations. The faster this rotation, in allowing agents that have previously not been rewarded to receive rewards, the greater the likelihood of envy-freeness.

7 Automated generation of diversity

In this section, we introduce the main contribution of this paper: a self-adaptation mechanism called *clonal plasticity* which can be used by a group of agents to self-diversify. We use Clonal Plasticity, to generate diversity automatically amongst the algorithms implemented. We show that this generated diversity is competitive with the manually introduced diversity presented in the previous section.

We had previously used clonal plasticity [101] to enable decentralized adaptation, based on localized environmental changes. Due to its decentralized nature, the adaptations performed by multiple agents in the system, results in variations being naturally produced at a system-wide level. The process of clonal plasticity is briefly explained in the following sub-section.

7.1 Clonal plasticity process

Clonal Plasticity is an adaptation strategy, formed through the combination of two processes present in plants: *(a) clonal reproduction*, and *(b) phenotypic plasticity*. *Clonal Reproduction* refers to "... sequential reiteration of a basic structural unit or module consisting of the leaf with its associated auxiliary bud and a segment of stem that connects it to other units" [102]. The fundamental idea in this form of reproduction is that the daughter plant is genetically identical to the parent plant, and hence the term *clone*. *Phenotypic Plasticity* refers to the ability of an organism to vary its structure (height of stem, width of leaves, etc.) or behaviour (connect to its clonal sibling or disconnect) according to environmental influences. Importantly, this adaptation does not include variation in an organism due to genetic factors. We combine these two mechanisms to propose a self-adaptation mechanism called *clonal plasticity*. Clonal Plasticity is distinguished from other evolutionary mechanisms such as *genetic algorithms*, in that the genome of the organism is **not** modified at all. From a software system's perspective, this means that a clonally plastic software will deviate in some aspects of its behaviour, but **will functionally remain the same**.

The process of plastic reproduction, depicted in Fig. 2, is given by the following steps:

1. *Identify plasticity points* At a given moment in time, t, of the system's lifetime, each individual agent in the system (depicted a circle in Fig. 2) identifies all its parts that can take multiple representations, values, or behaviour.
2. *Evaluate environmental input* Simultaneously, the fitness of each individual agent is self-evaluated in the following way: for each plastic point, environmental feedback is evaluated to check whether it impedes or favours that point. Feedback from the environment may come from different sensors or neighbouring agents, or the agent's own reproduction memory (m in Fig. 2).

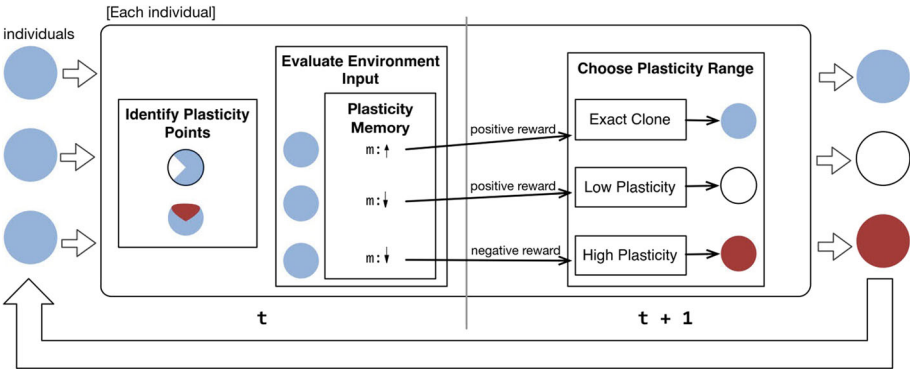

Fig. 2 Clonal plasticity process

3. *Plasticity memory* All agents that have reproduced before, keep a record of the previous Plasticity Range chosen during the last reproduction cycle (variable `m` in Fig. 2). This allows an individual agent to repeat a good adaptation strategy, if it previously had a positive reward, or alternatively abandon a poor adaptation strategy.
4. *Choose plasticity range* During the following time-step (`t+1`), each agent chooses a plastic response, based on the evaluated environmental input. The available responses may be one of:

   (a) Exact Clone,
   (b) Low Plasticity, and
   (c) High Plasticity.

5. *Clone and modify plastic points* During the same time-step, each agent makes a clone of itself. The individual agent's plasticity points may be adapted according to the chosen strategy. The entire process starts again with a new time-step, t', on all clones (Fig. 3).

7.1.1 Cloning frequency

Every adaptation action, in any system, incurs a cost, either computational or communicative. It must be possible to evaluate the effect of the cloning process, before creating the next plastic clone. `t + 1` refers to the next time-step when it is feasible to perform the next cloning process. In the agent that implements plasticity, the granularity of time when the next iteration of the cloning process is done, is domain dependent. For instance, domains such as traffic and smart-grids are different in how dynamically their respective environments change. To allow for each adaptation to take effect, we introduce the notion of *cloning frequency*. Cloning Frequency refers to the time-period for which an agent is evaluated before the next iteration of the cloning process starts.

7.1.2 Choosing the plasticity range

The process of choosing the plasticity range is the critical part in generating a diverse population. This process of modifying the plastic points considers both positive and negative feedback from the environment. For a given time instant `t`, the adaptation decision depends on two factors:

```
Data: PlasticSet, Memory, LastReward
Result: AlgorithmClone
Algorithm CreateClone(PlasticSet, Memory)
    foreach Agent a do
        LastReward_a ← EvalEnviron(a, PlasticSet_a)
        WhichRange_a ← ChoosePlasticityRange(LastReward_a, Memory_a)
        switch WhichRange_a do
            case Exact-Clone do
                clone ← a
                return clone
            case Low-Plastic-Clone do
                clone ← a
                plasticity − point, direction ← Memory_a
                ModifyAgent(clone, plasticity − point, direction)
                return clone
            case High-Plastic-Clone do
                clone ← a
                last-plasticity-point, last-direction ← Memory_a
                plasticity-point ← chooseDifferent(PlasticSet_a, last-plasticity-point)
                direction ← chooseDifferent(last − direction)
                ModifyAgent(clone, plasticity − point, direction)
                return clone
Algorithm ChoosePlasticityRange(LastReward, Memory)
    if LastReward < 0 then
        return Low-Plastic-Clone
    else
        if Memory > 0 then
            return Exact-Clone
        else
            return High-Plastic-Clone
Algorithm EvalEnviron(Agent a)
Algorithm ModifyAgent(Agent a, plasticity − point, direction)
```

Fig. 3 Create a plastic clone of an algorithm

1. Feedback from environment at time `t`, whether it is positive or negative.
2. Memory of the modification action, taken at time `t-1`.

If the feedback from the environment is positive, and memory of the previous action is also positive, then the Plasticity Range is chosen to be *Exact Clone*. If the feedback from the environment is positive, but the memory of the previous action is negative, then the Plasticity Range is chosen to be *Low Plasticity*. If the feedback from the environment is negative, then the Plasticity Range is always *High Plasticity*

7.1.3 Exact clone

In this case, the individual simply makes a copy of itself, with no change at all.

7.1.4 Low plasticity

Modification with Low Plasticity implies that the individual chooses the same plasticity point, as chosen during the previous cloning process, and moves in the same direction as before.

7.1.5 High plasticity

Modification with High Plasticity implies that the individual chooses a different plasticity point, than was chosen during the previous cloning process, and moves in a random direction.

7.2 Clonal plasticity for MG

Here, we modify the agents, such that each agent is able to reproduce in a clonally plastic way. Cloning, as opposed to other evolution-based mechanisms, is completely decentralized as there is no universal fitness function that sorts the population in to *fit* and *unfit* individuals.

Making an algorithm plastic requires following the process of plastic reproduction

1. *Identification of plasticity points* BestPlay has only one variable that influences its decision-making, *i.e.,* `memory`. Hence, there is only one plasticity point for agents implementing BestPlay. Similarly, for the Evolutionary strategy, there is one variable that influences how the decision making varies within the population, *i.e.,* `mutation`. Roth-Erev has two variables that can be plasticized, *i.e.,* `recency` and `epsilon`
2. *Evaluate environmental input* The environment allows each agent to know how well its algorithm is doing. That is, depending on the performance of the agent in successfully choosing the minority group, the agent would have either get positive or negative feedback. In the context of MG, cloning frequency refers to the number of times an agent plays the game, before its performance is evaluated. Thus, a cloning frequency of 5 means that an agent's performance is evaluated after every 5 rounds. Clearly, the lower this number, the faster the agent reacts to the environmental feedback. Agents playing BestPlay or Roth-Erev were evaluated with cloning frequency ranging from 5 to 105 rounds. Agents playing the Evolutionary strategy were evaluated with cloning frequency ranging from 20 to 100. This difference is due to the fact that preliminary experimentation revealed that the Evolutionary algorithm required 20 rounds to produce a noticeable effect in an agent's performance. The effect of cloning frequency on diversity generation is discussed in Sect. 9.2. The memory (m) of the previous algorithm parameters allows an agent to return to a previously known good configuration, if the current modification resulted in losses.
3. *Choose plasticity range* The plasticity range chosen by a particular agent defines how the agent decides to adapt. Each of the algorithms (BestPlay, Evolutionary, Roth-Erev), depending on its choice of parameters, decides how the agent would play upcoming rounds of MG. Any change of these parameters could cause the agent to change its decision, and thereby affect the rewards that it accumulates.

Table 3 Criteria for choosing plasticity range

Feedback	Plasticity memory	Plasticity range chosen
Negative	(Disregarded)	High plasticity
Positive	wins-this-cycle <= wins-last-cycle	Low plasticity
Positive	wins-this-cycle > wins-last-cycle	Exact clone

7.3 Worked out example

Now we perform the clonal plasticity process step-by-step for BestPlay. For details on the clonal plasticity process for the Evolutionary and RothErev algorithms, see appendices A and B (the process differs only in the plasticity points identified for each algorithm).

1. *Identify plasticity points* The BestPlay algorithm critically depends on the size of the memory, that it keeps of the game (see Table 1). Two agents that diversify into different directions (one increasing the memory size, and the other decreasing) could end up with drastically different rewards. Here, m is identified as a plasticity point of the algorithm. Another plasticity point is the number of strategies (k) that the agent possesses.
2. *Evaluate environmental input* For any given time instant, the algorithm has a 50% chance of picking the right group (*i.e.*, being in the minority). Hence, for any given period of cloning frequency, if the agent has accumulated (`wins-this-cycle`) more than 50% of the rewards during the period, the feedback is said to be positive. If the agent accumulates less than 50% of the rewards, then the feedback is negative.
3. *Plasticity memory* The agent keeps track of the rewards that it had accumulated during the previous adaptation (`wins-last-cycle`). Along with the feedback, this helps it decide what plasticity range to choose next (see Table 3).
4. *Clone and modify plastic points* In the event of *Exact Clone*, the agent does not change any of its parameters. In the event of *Low Plasticity*, the agent picks one of its plastic points and modifies it in the same direction, as the plasticity memory from the previous generation. In the event of *High Plasticity*, the agent modifies *all* its plasticity points to return to the values that it had from the previous generation.

8 Requirements for distributive justice

In Sect. 5.1, we saw that efficiency in allocations would only occur, if $minG = N/2$. However, it is entirely possible that in repeated iterations of the game, the same subset of N agents continue to be a part of $minG$. This would mean that the allocations while efficient, are unfair, leading to envy amongst the agents. Ideally, we would like *envy-freeness* to be a property of clonal plasticity. While *envy-freeness* is an important concern in many domains, there are some areas, specifically in simultaneous games, where envy-freeness is impossible [16]. Balkanski *et al.* [16] show that for any resource allocation protocol which involves simultaneous choice by more than 3 agents, it is impossible to construct an envy-free protocol. This impossibility is intuitive in the case of the Minority Game where, by construction, during every round, a majority of the agents are denied a reward and are envious of the minority.

In view of the impossibility of an envy-free protocol ever being created, we focus on an alternate form of fairness, *i.e., distributive justice.*

***Distributive justice*, in the context of MG, implies the following condition**

$$\sum_{t=1}^{T} R_t(a_j) \approx \sum_{t=1}^{T} R_t(a_k) \forall (a_k \neq a_j) \tag{2}$$

Equation 2 simply states that, over a period T, every agent (a_j) in MG must get approximately the same reward $(R(a_j))$, as any other agent (a_k). In the ideal case, every agent would get **exactly** the same reward as every other agent. However, this can be trivially shown to be impossible under the following conditions of agent abilities:

1. *Autonomy* Each agent is autonomous and cannot be coerced by any other agent. The agent's choice is dependent solely on its own algorithm.
2. *Simultaneity* Each agent makes its choice in a simultaneous fashion, and no agent has access to any other agent's decision before it makes its own.

Proof
Assume a set of algorithms A that are implemented in the agents playing MG. If this set A meets two conditions, then MG would be both, efficient and fair:

1. The size of the minority is the largest possible, while still being a minority, and
2. The members of the minority group, in any iteration of MG, have never been in a minority before

then, after two iterations, there will be only one agent that has never been in a minority before. Therefore, in iteration 3, the sole agent participates in the minority group, and thus meets the game is both, efficient and fair.

For any mechanism, meeting condition 1 requires that the size of the $minG$ be $N/2$ for iteration 1 **and** iteration 2. Meeting condition 2 requires that the list of agents that have already participated in $minG$ be known at all times, and any agent that has previously been a part of $minG$ be restricted by some mechanism. Meeting condition 2 violates the environmental assumption of *Autonomy*, while meeting condition 1 violates our environmental assumption of *Simultaneity*. Thus, there is no way for any mechanism to assure equality of rewards and be efficient, without violating our operating conditions.

8.1 Desirable properties of clonal plasticity

It is evident that equation 2 would be impossible to achieve with MG, if it were played in a one-shot manner. That is, for a single round of play, regardless of the algorithm chosen, the agents that are in the minority will get a reward, while the agents in the majority will not. This means that equation 2 will never hold. However, if the same agents play MG in a repeated manner, it is probable that different agents are in a minority in other rounds, such that their rewards accumulated over time would be approximately equal. That is:

$$\lim_{T \to \infty} \sum_{t=1}^{T} R_t(a_j) \approx \sum_{t=1}^{T} R_t(a_k) \forall (a_j \neq a_k) \tag{3}$$

Recall, that we are not interested solely in distributive justice, but also that the mechanism should be efficient. In the case of MG, the system would be efficient if allocation of rewards is

Table 4 Experimental constants–Same as manual process of diversification

Variable	Value
Population size	501
Simulation Period (rounds)	2000
Repetition of variation	100

as high as possible in any given round (see Sect. 5.1). At its most efficient, the game provides the highest amount of reward (when the size of $minG$ is $N/2$):

$$R_1 = \frac{N}{2} \tag{4}$$

This implies that after T rounds of highest possible efficiency, the total rewards in the system must be:

$$\sum_{t=1}^{T} R_t = \frac{N}{2} * T \tag{5}$$

From equations 4 and 5, it is clear that at optimal efficiency, the expected rewards accumulated by any single agent is given by:

$$\begin{aligned} \sum_{t=1}^{T} R_t(a_j) &= \frac{N/2 * T}{N} \\ &= \frac{T}{2} \end{aligned} \tag{6}$$

Thus, we would like a mechanism to achieve **both** equation 3 and 6.

9 Experimental setup and results

Table 4 shows the constant factors in each experiment.[3] Each minority game was played with a population size of 501 agents, through a simulation time period of 2000 steps. For each variation in the experimental setup, the data is reported as an average of 100 simulations. We measure the efficiency of the system through the median amount of rewards accumulated by the agent, after the simulation period. We measure the fairness of the system using the Gini index. The Gini index is used to calculate dispersion in the income distribution of a society [103]. In a society that is perfectly equal, the Gini index is equal to zero. Therefore, the closer the Gini index is to zero, the fairer the distribution of rewards.

Figures 4 and 5 show the results of diversification introduced by clonal plasticity. As is immediately obvious from the x-axis (that shows diversity generated), manual diversification is able to generate a higher level of diversity in the populations of agents. Clonal Plasticity is able to reach a diversity level (also called H-index) of upto 2.7, whereas manual diversification (see Fig. 1) is able to reach an H-index of upto 5.8. This is due to the fact that a human (knowing the functional pathways of a particular algorithm) is able to specify diverse individuals that will generate diverse outputs. However, in terms of efficiency and equity, clonal plasticity confirms the results found by *Nallur et al.* [19]. Note that as the diversity goes up, the median

[3] All code for this experiment can be found at: https://bitbucket.org/viveknallur/clonal_plasticity_algo_diversity.git.

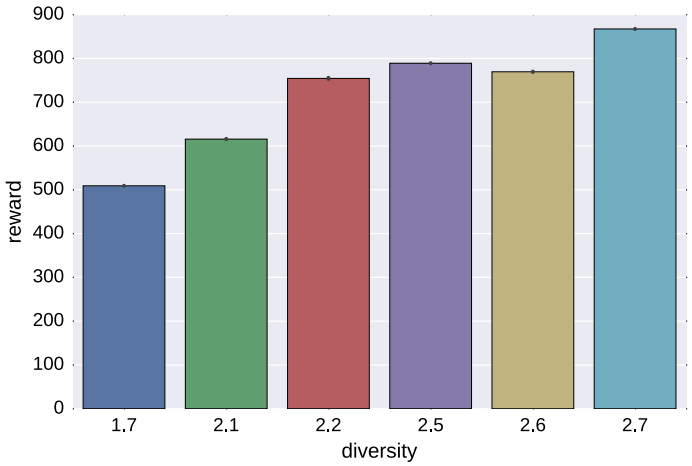

Fig. 4 Impact of diversity on reward

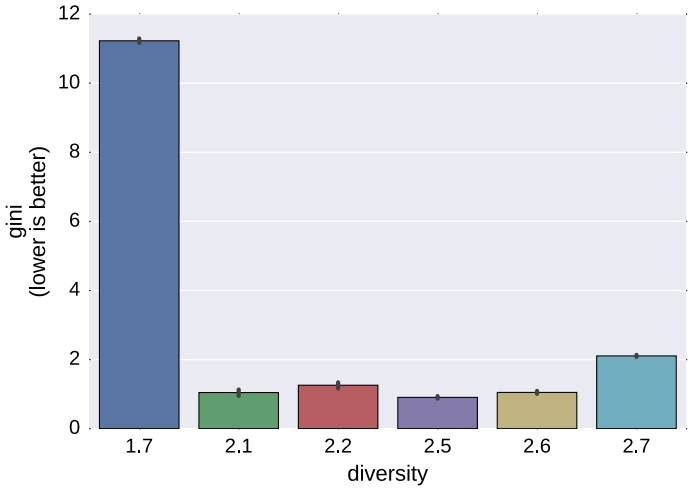

Fig. 5 Impact of diversity on fairness

reward accummulated by the population also increases (Fig. 4). At the lower end of the diversity scale ($H = 1.7$), the median reward is 509, while at the higher end of the scale ($H = 2.7$), the median reward is 867. Increasing the diversity of the population increases the reward achieved by each agent. Figure 5 shows the effect on equity of rewards achieved. The *y-axis* shows the Gini index, which is a quantitative measure of equality of reward. In Fig. 5, the increase in diversity results in a decrease in the Gini index (and therefore, a fairer distribution). *Note:* Both Figs. 4 and 5 are shown with error bars that indicate the 95% confidence intervals. The medians were established using 100 experimental trials for each variation (7 variations in algorithms [see Table 6]), and the confidence intervals for each diversity value across the 700 trials were established by bootstrapping at 100 iterations.

Table 5 Relationship between diversity and reward and fairness

Diversity	Reward	Gini
1.7	509	11.2
2.1	616	1.0
2.2	755	1.3
2.5	789	0.9
2.6	770	1.1
2.7	867	2.1

9.1 Effectiveness of diversity in fairness and efficiency

Recall from Sect. 8, the theoretical optimum efficiency that we would like is $T/2$. That is, the accummulated reward at the end of the simulation should be equal to half the number of rounds of simulation. In all our experiments, the T is kept as a constant 2000, hence the optimum efficiency is 1000. We see from Fig. 4, that increasing diversity results in a increased median reward. At the highest diversity level (H = 2.7), the accumulated reward is 867, while at the lowest level (H = 1.7), it is 509.

Note also from Table 5, the lowest level of diversity is also responsible for the highest level of the Gini index. Recall that the Gini index measures the inequality of the distribution of income. Thus, the closer the Gini index value is to zero, the more equitable the distribution is. This table shows that attaining both, efficiency of the mechanism (close to the theoretical optimum) and fairness (a theoretical optimum of zero) is a difficult problem. While increasing diversity directly increases the efficiency (867 is closer to 1000 than 509), it does not have the same clear impact on fairness. While the fairness achieved with the highest diversity (H-index of 2.7 results in Gini index of 2.1) is certainly better than the fairness achieved with low diversity (H-index of 1.7 results in Gini index of 11.2), it is interesting to see that a diversity level of 2.5 actually provides the highest amount of fairness (Gini index of 0.9). This indicates the system might have an optimal 'sweet spot' for diversity, and going beyond such a limit might not necessarily improve both efficiency and fairness. Automatically deriving the pareto curve of diversity, for a particular system, would be an interesting avenue of further research.

9.2 Evaluating clonal plasticity in generation of diversity

We now look at Clonal Plasticity itself and evaluate how the cloning frequency influences the amount of diversity achieved within the population. Cloning frequency refers to how many rounds (of time) elapse before the cloning process is run. Intuitively, the fewer the number of rounds between each cloning process, the higher the amount of diversity that will be achieved. This is due to the fact that each run of the cloning process has the potential to create a new specie. This is borne out by the data in Table 6 which illustrates that an increase in mean diversity has a direct correspondence with an increase in the mean number of species generated by the cloning process. Figure 6 shows a population-wide distribution of the same analysis

In Fig. 6, we see the distribution of diversity across the entire population as bubbles (*y-axis*), with change in cloning frequency(*x-axis*). The size of the bubbles represents the number of species. As we go up the *y-axis*, notice that the size of the bubbles increases. This indicates that the greater the diversity, the greater the number of species generated. Also, the lower the

Table 6 Diversity and number of species

Algorithm	Diversity	Num_Species
BestPlay	1.7	60
Evo	2.7	501
RothErev	2.7	486
BestPlay-Evo	2.1	271
RothErev-BestPlay	2.2	303
RothErev-Evo	2.7	493
Evo-RothErev-BestPlay	2.5	419

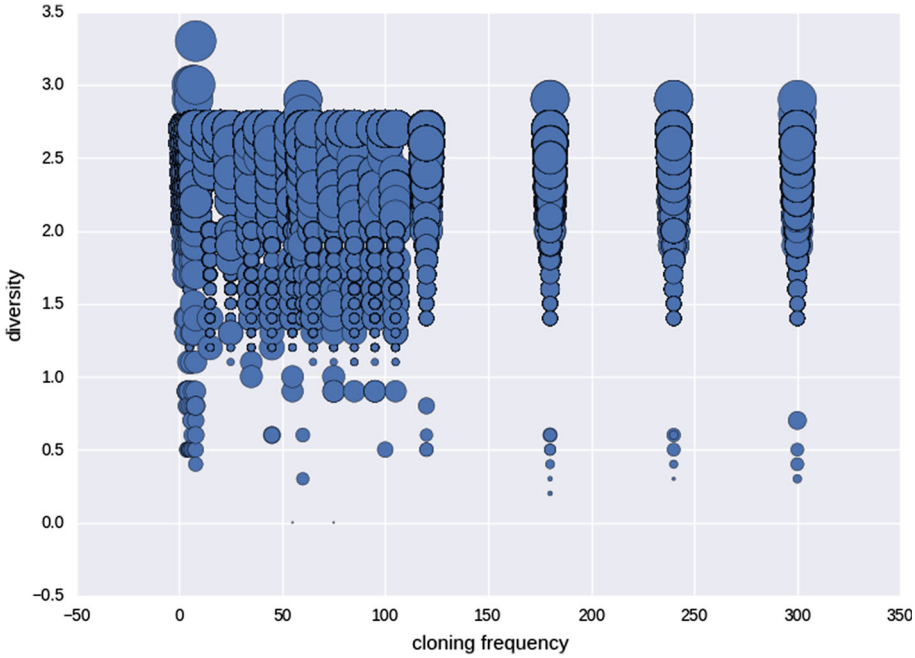

Fig. 6 Diversity, cloning frequency and number of species together

number of rounds between each cloning process (*i.e.,* the lower the frequency), the higher the mean amount of diversity produced. Table 9 in Appendix C shows the full breakdown of the number of species produced, as the cloning frequency varies, per algorithm. It is interesting to note that the cloning frequency does not, by and large, affect the diversity achieved *within* an algorithmic group.

9.3 Effect of plasticity on different algorithms

Figures 7 and 8 show an algorithm-wise breakup of differences in reward achieved as well as the Gini index. Read in conjunction with Fig. 9, which shows that three algorithms (`Evo`, `RothErev` and `RothErev-Evo`) achieved the highest diversity, it also shows that high levels of diversity are linked to high levels of reward. Table 9, from Appendix C shows how this can be further broken down based on how rapidly the cloning process operates.

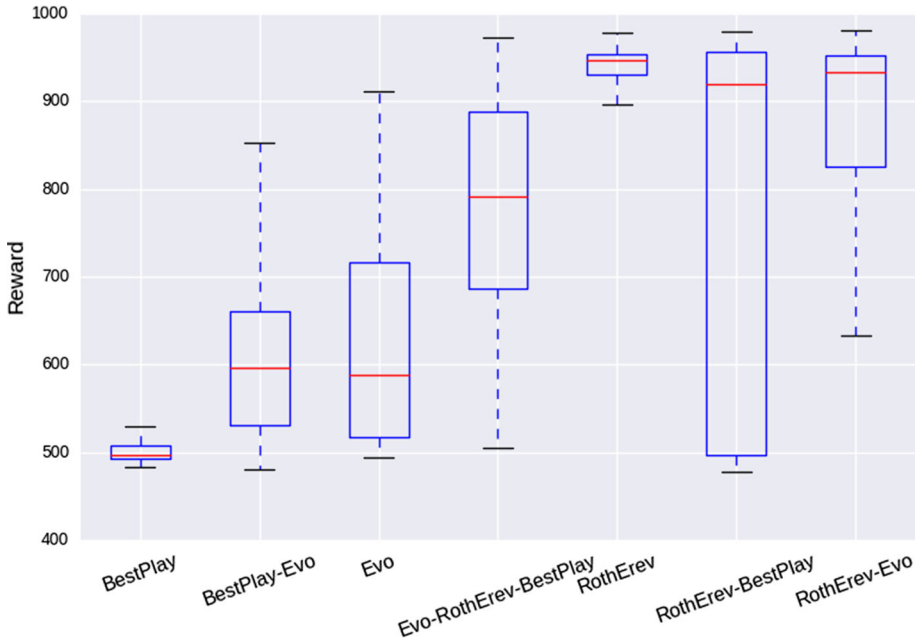

Fig. 7 Algorithm-wise breakup of reward achieved

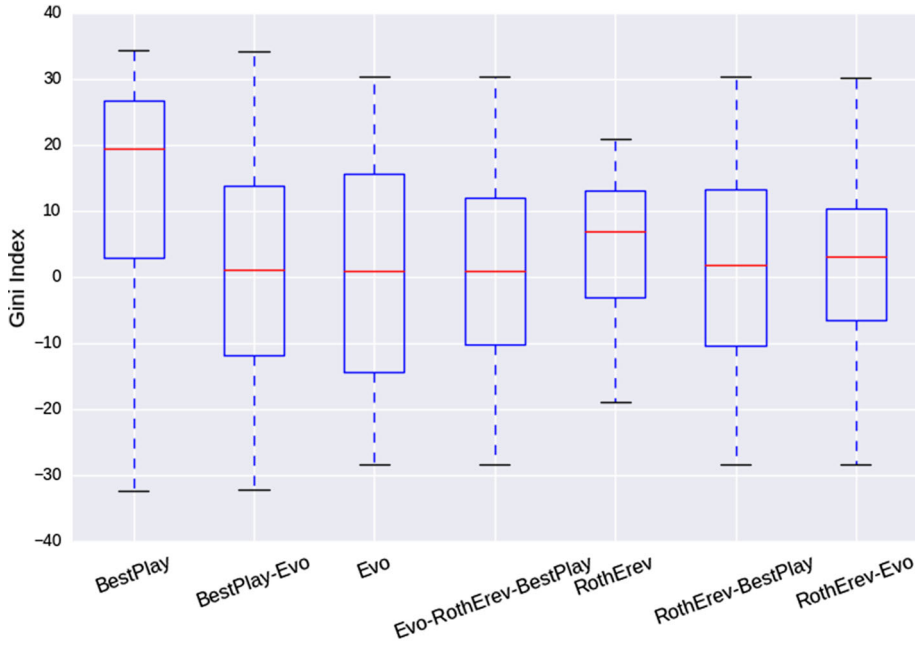

Fig. 8 Algorithm-wise breakup of Gini Index

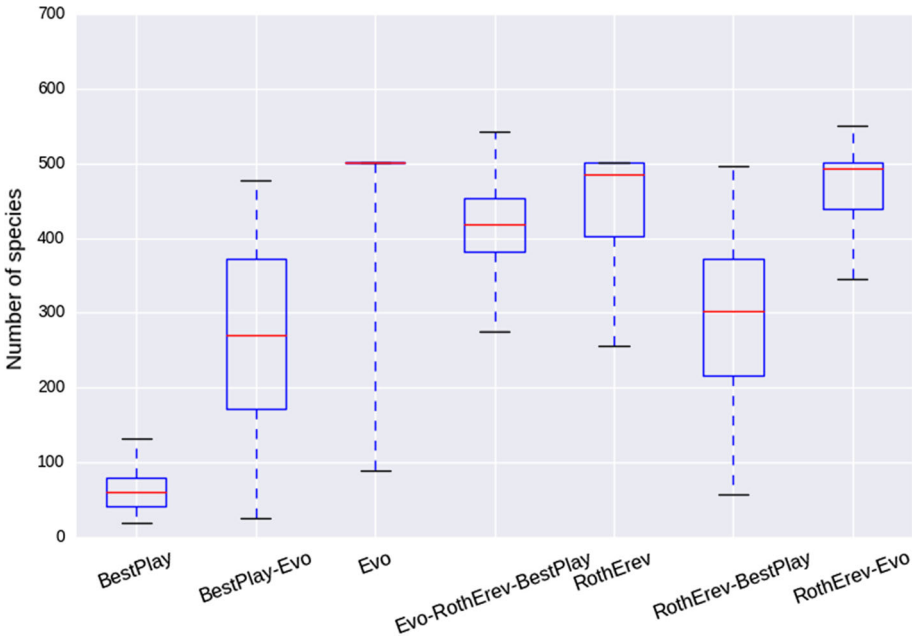

Fig. 9 Algorithm-wise breakup of number of species at the end of cloning

Consider Figs. 7 and 8 along with Tables 5 and 6. Both `BestPlay-Evo` and `RothErev-BestPlay` achieve a mean Gini-index very close to zero(1.0 and 1.3), however it is the combination of `Evo-RothErev-BestPlay` that achieves the best Gini-index (0.9) and as well as a very high level of mean reward (780). `Evo-RothErev-BestPlay` performs very well across both indicators of reward and Gini. However, if one considers only rewards, it is `RothErev` with the highest diversity level of 2.7, that has the highest reward level. In contrast, `Evo` is another algorithm that reached a high level of diversity; almost all its individuals consistently formed separate species (see Fig. 9) and a mean Gini-index very close to zero. However, an examination of Fig. 8 shows that the distribution of Gini values fluctuated quite a bit (see Table 9 from Appendix C for a cloning-frequency-wise breakdown), indicating that it is sensitive to cloning-frequency values. Taken together, the results show that raising diversity levels indiscriminately is not a silver bullet for achieving fairness, however reward levels rise in congruence with diversity.

9.4 Scalability of clonal plasticity

Although the clonal plasticity process is effective at generating diversity, its scalability is an important factor in deciding whether it is feasible for implementation in real systems.

Before the cloning process can begin, the identification of the plasticity points must be done. This is a one-time function and needs to be done only at the initialization of the plasticity process. For any algorithm `a`, with `k` interface variables, the first step is to identify whether those variables modify the output of the algorithm. This is clearly $O(k)$ in complexity. The second major step is to identify the plasticity range for each plasticity point. This involves identifying the valid set of values a plasticity point may take, as well as the constraints on

its value. Assuming a time complexity of r per variable, the first phase of clonal plasticity is $O(r * k)$ in complexity.

Recall that the `ChoosePlasticityRange` and `CreateClone` functions form the core of the plasticity process (see Algorithm 3). `ChoosePlasticityRange` is a linear function ($O(c)$) since it involves merely decision-making, based on the `LastReward` and `Memory` variables. `CreateClone` involves looking at the value of each plasticity point, and assuming a time complexity of p per plasticity point, its complexity can be calculated as $O(p * k)$. The computational complexity of the clonal plasticity process is therefore: $O(r * k) + O(c) + O(p * k)$. This is important because, if the adaptation process is computationally expensive, then its benefits are limited to systems with few plasticity points. The linear complexity of clonal plasticity implies that the mechanism is scalable to large-scale multi-agent systems, with a high number of plasticity points.

The `EvalEnviron` is a domain-specific function to evaluate the current performance of the agent, and is therefore left unspecified. Any adaptation mechanism would require an `EvalEnviron` function and as such, it is a price that all adaptation functions must pay for sensible self-modification.

Crucially, it is important to note that the entire process works on a single agent and is independent of other agents' performance, in any direct fashion. This allows the plasticity mechanism to work in a completely decentralized manner. Of course, the presence of other agents affects the evaluation of the environment (in an indirect fashion via their actions), and thus provides local competitive pressure that moves the agent to a better configuration.

10 Differences between clonal plasticity and phenotypic diversity-based mechanisms

In the presence of large search spaces for adaptation, there have been a number of *intelligent trial-and-error* based algorithms that have been proposed. These differ from the more commonly known EA/GA/GP methods in that there is an explicit absence of an optimzing function. Notable examples include techniques such as Novelty Search [104], Functional Blueprints [105] and more recently, MAP-Elites [106], SHINE [107], etc. Among these techniques, there is an acknowledgement of the fundamental importance of phenotypic diversity, similar to clonal plasticity. The key difference between these methods and clonal plasticity is *pre-computation* of phenotypically diverse behaviours. For instance, Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [106] creates a number of discrete behaviour bins, which contain the fittest individual mapped to that bin so far. The bins are then sampled at random to produce an offspring and mapped to a bin depending on its behaviour characterization. The individual is then saved or culled depending on its relative fitness to other individuals in the bin. In this way, due to uniform sampling, the algorithm passively acquires diversity [108]. SHINE [107] uses the same fundamental idea, but uses a tree-structure for maintenance and selection of potential individuals. This, along with a technique of penalising crowded areas, allows the algorithm to explore the phenotypic landscape more rapidly than MAP-Elites.

Real-world domains that have been explored by these techniques include bi-pedal walking by a robot, walking by hexapod robot and damaged robotic arms. A peculiarity of these domains is that the adaptation required cannot wait for *trial-and-error* to conclude, and

hence pre-computation is a necessity for such domains. Pre-computation requires simulation and a thorough exploration of the behaviour space, before the final solutions are transferred to the hardware device. Clonal Plasticity does not do any pre-computation and hence is unsuitable for domains that require a rapid response.

11 Differences between clonal plasticity and non-phenotypic diversity-based mechanisms

Clonal Plasticity is a meta-heuristic, in the same vein as the other methods in evolutionary computation. However, it differs in significant ways from the diversity generation approaches presented before, in Sect. 3. These axes affect how suitable an approach is (Table 7), for a specific domain:

1. **Representation** (Neutral or Dependent): If the diversification relies on the distance-measure adopted, then the representation of the individual becomes a critical factor in calculating diversity.
2. **Temporal Dependence** (Yes or No): Does the diversification depend on *when* the differences between individuals is calculated. For example, methods such as *incest prevention* and *mating restrictions* require that the diversity between individuals be calculated **before** every new generation. Similarly, the ensemble mechanism also requires that different values for ensemble training are chosen (from a poisson distribution) before the training starts.
3. **Locus of Control** (Yes or No): Does the mechanism involve calculating the diversity of the entire population in a centralized fashion or can different subsets diversify at different rates?

11.1 Representation

Evolutionary Algorithms (EAs), Genetic Algorithms (GAs), Genetic Programming (GP) typically use a distance measure in the genotype of the individual to force diversity into the population. For example, Sareni and Krähenbül utilize the notion of fitness sharing and creation of niches to ensure that the population retains diversity, after multiple generations [109]. However, this mechanism quickly becomes computationally intractable when using genotype representations such as Artificial Neural Networks(ANNs) [110]. Behavioural diversity based approaches such as novelty search [104] avoid the problems of genotype-based representations.

The traditional particle-swarm [75] uses a tuple of D-dimensional vectors (where D is the dimensionality of the search space), position and fitness of best point, as well as index of best particle to represent a particle in the swarm. Diversity generation mechanisms (*e.g.*, [79,80]) continue to use the tuple as a standard representation of a particle.

Machine Learning approaches such as ensembles are independent of any particular representation.

Clonal Plasticity does not have (or require) a particular representation of an agent. The agent must merely be able to reflect on, and tune its own parameters.

Representation is an important aspect during development, since many application domains where multi-agent systems are used (modelling electricity grids, transportation systems, etc.) do not lend themselves to arbitrary representations. Therefore, for agents that

need to model domain-specific behaviour it would be more convenient to employ either ensemble-based diversity generation or clonal plasticity.

11.2 Temporal dependence

EA/GA/GP have a high dependence on the temporality of the diversity calculation. Due to their inherent selection-pressure based nature, diversity mechanisms must be deployed at every generation, to ensure that the selection pressure does not lead to a homogeneous/convergent population.

Swarm-based methods are also dependent on temporality of diversity calculation, since every particle's next-step will influence whether the resulting swarm consists of a dense neighbourhood(s) or not.

Machine Learning mechanisms such as ensemble-learning are not dependent on generations, however diversity is induced during the training phase by using variation in the examples [37]Ḣowever, other mechanisms such as evolutionary modification of neural networks [81] require a generation-by-generation diversity calculation.

Clonal Plasticity belongs to a class of methods which do not explicitly force selective pressure towards diversity [111], rather diversity is emergent from the self-adaptation behaviour.

The need to ensure diversity at every generation makes EA/GA/GP/swarm-based approaches unsuitable for a dynamic environment that requires some action from the system-under-consideration. Unlike optimization problems where the final calculated solution is the only determiner of quality, other domains (such as human-facing systems) will often need software to respond, even if sub-optimally. In such situations, clonal plasticity is the best way to ensure that the system continues to function, while moving towards more diversity.

11.3 Locus of control

EA/GA/GP use methods that calculate genotype-distance require the entire population to be available for diversity calculations. Even the behaviour-based novelty search creates a distance-measure among behaviours that have been observed in the population, and hence require computation over the entire population.

Swarm-based methods operate on each particle at a time, however, neighbourhood measurements of density require computations over subsets of the population, which make diversification more of a centralized process than a decentralized one.

Machine Learning techniques all explicitly aim for diversity during their training phase. This requires that after a training datum, the feedback and subsequent training take the current homogeneity of all available learners into account.

Since clonal plasticity does not explicitly aim towards diversity, the diversity seen in the population is an emergent phenomenon from the actions of the individual agents. Each agent does, however, perform a neighbourhood-based calculation of its relative fitness. This makes clonal plasticity not a fully decentralized process, rather a distributed process.

No approach that we know of is able to achieve a fully decentralized operation to ensure diversity. The best achieved (so far) reduces computation to small subsets, instead of the entire population. Decentralization is required where the system-under-consideration is a large-scale system, and using centralized operations are infeasible. Examples of such systems include vehicular ad-hoc networks [3,4], smart buildings [5], e-procurement [6], cloud computing [7], healthcare systems [8].

Table 7 Contrast between diversity generation approaches

	EA / GA / GP	Swarm-based approaches	Machine learning	Clonal plasticity
Representation	Distance measure based on genotype	Tuple-based representation of particle	Independent	Independent
Temporal dependence	Diversity is calculated after every generation	Diversity is calculated after every generation	Ensemble-learning dependent on training phase	Not dependent
Locus of control	Centralized	Centralized	Centralized	Distributed

12 Weaknesses of clonal plasticity

Domain limitations Clonal Plasticity is an intelligent trial-and-error mechanism that is able to dynamically adjust to a changing environment by exhibiting phenotypic diversity. However, in certain domains such as robotics or autonomous vehicles the rate of response required from the system is quite fast. Due to real-time response requirements, other phenotypic diversity mechanisms such as MAP-Elites [106] use *pre-computation* to explore the possible behaviour space before deployment. Clonal Plasticity does not use pre-computation and hence is slow to jump from one known behaviour to another. Domains requiring such fast adaptation responses medical devices, transportation, and industrial robotics are not suitable for clonal plasticity.

Algorithmic limitations Clonal plasticity is a meta-algorithm that achieves diversity regardless of the underlying problem-specific algorithm being implemented. However, the diversity-seeking behaviour makes it unlikely that clonal plasticity can be used to achieve optimality in any search space. This diversity, unlike others such as Novelty Search [104] and SHINE [107], is an emergent characteristic of the interaction between plasticity and the environment. This implies that if by some chance, clonal plasticity hits on the optimal behaviour, it is likely to stay at that point. However, it does not actively seek out optimal behaviour. By contrast, EA/GA/GP have a well-established record of optimization with both, theoretical analysis [112] as well as in practical domains [113,114]. Hence, clonal plasticity is unsuited for situations that require optimization.

13 Conclusion

In ecological sciences, as well as computer science, diversity has been shown to be an important property of an ecosystem. As pointed out in Sect. 2, the concept of diversity has been used in computer security, search, machine-learning, as well as software engineering. However, instead of qualitative statements about diversity, we present a quantitative approach to measuring the effects of diversity, and an adaptation approach that increases the amount of diversity present in the system. This is especially important in socio-technical systems that involve human interactions. The *Minority Game* has been shown to be a simple, yet good proxy for socio-technical systems such as traffic [115], financial markets [93], distributed decision-making in wireless networks [94], smart-buildings [95], etc. Results shown in the *Minority Game* setting indicate that diversity can play an important role in affecting system

properties such as efficiency and fairness. We do not wish to suggest that diversity is a silver bullet for achieving optimal levels of both, but rather that it is an important mechanism in a software engineer's toolkit while engineering socio-technical systems.

13.1 Open research questions

Although clonal plasticity has been shown to lead to successful diversification, it would be interesting to know if there are limits to how quickly such diversification can occur. An investigation into the formal properties of the clonal plasticity process would be invaluable in determining its suitability for time-critical systems. Are there properties of the constituent algorithms or the number of plasticity points that can predict the speed of, or magnitude of adaptive movement? Can we automatically identify the pareto space of the system, where increasing diversity does not necessarily result in better values of the desired system properties? All of these are subjects of future research that we think will contribute greatly to a more detailed and nuanced understanding of this adaptive strategy.

13.2 Concluding remarks

Multi-agent systems are increasingly being used to model complex socio-technical systems that involve human interactions. In such systems, it is difficult to simultaneously guarantee properties of efficiency and fairness. This paper presents a measurable system property (diversity) that can be engineered to produce high levels of both. Further, it presents an adaptation process (clonal plasticity) that can drive a system, from low levels of diversity to higher levels, depending on the number of plasticity points that individual agents possess. Being a completely decentralized mechanism, clonal plasticity is suitable for systems consisting of a large number of agents.

Appendix A

The clonal plasticity process step-by-step for *Evo* is as follows:

1. *Identify Plasticity Points* The Evo algorithm diversifies itself based on the mutation probability of an individual. Two agents that diversify into different directions (one increasing the mutation probability, and the other decreasing) could end up with drastically different strategies after their evolutionary reproduction. Here, $ownMutationRate$ is identified as a plasticity point of the algorithm.
2. *Evaluate Environmental Input* For any given time instant, the algorithm has a 50% chance of picking the right group (*i.e.*, being in the minority). Hence, for any given period of cloning frequency, if the agent has accumulated (`wins-this-cycle`) more than 50% of the rewards during the period, the feedback is said to be positive. If the agent accumulates less than 50% of the rewards, then the feedback is negative.
3. *Plasticity Memory* The agent keeps track of the rewards that it had accumulated during the previous adaptation (`wins-last-cycle`). Along with the feedback, this helps it decide what plasticity range to choose next (see Table 8).
4. *Clone and Modify Plastic Points* In the event of *Exact Clone*, the agent does not change its plasticity point. In the event of *Low Plasticity*, the agent modifies its $ownMutationRate$ in the same direction, as the plasticity memory from the previous generation. In the event of *High Plasticity*, the agent modifies its $ownMutationRate$ to return to the value that it had from the previous generation.

Table 8 Criteria for choosing plasticity range

Feedback	Plasticity memory	Plasticity range chosen
Negative	(Disregarded)	High plasticity
Positive	wins-this-cycle <= wins-last-cycle	Low plasticity
Positive	wins-this-cycle > wins-last-cycle	Exact clone

Appendix B

The clonal plasticity process step-by-step for *RothErev* is as follows:

1. *Identify Plasticity Points* The RothErev algorithm depends on two variables, $Recency$ and $Epsilon$. These affect how much weight the algorithm gives to the recent past, and how much exploration it does, respectively. Any difference in the values of these variables can cause agents that start out the same, to diverge in their strategies.
2. *Evaluate Environmental Input* For any given time instant, the algorithm has a 50% chance of picking the right group (*i.e.*, being in the minority). Hence, for any given period of cloning frequency, if the agent has accumulated (`wins-this-cycle`) more than 50% of the rewards during the period, the feedback is said to be positive. If the agent accumulates less than 50% of the rewards, then the feedback is negative.
3. *Plasticity Memory* The agent keeps track of the rewards that it had accumulated during the previous adaptation (`wins-last-cycle`). Along with the feedback, this helps it decide what plasticity range to choose next (see Table 8).
4. *Clone and Modify Plastic Points* In the event of *Exact Clone*, the agent does not change any of its parameters. In the event of *Low Plasticity*, the agent picks one of its plastic points and modifies it in the same direction, as the plasticity memory from the previous generation. In the event of *High Plasticity*, the agent modifies *all* its plasticity points to return to the values that it had from the previous generation.

Note that in this paper, we have differentiated the algorithms based only on their plasticity points. It is also possible to modify the rules by which the environmental input is evaluated and how the plasticity range is chosen, for each individual algorithm.

Appendix C

In this section, we show detailed information about how cloning frequency affects both, the diversity in the population as well as the *Gini* index. Recall that the Gini index measures the equity of reward in a society. The closer the absolute value of the index is to zero, the more equitable a society is. Conversely, the higher the Gini index, the less equitable distribution of rewards in that society. Table 9 shows the mean diversity, rewards and gini index for each cloning frequency, for each mix of algorithm. The final column (`std dev`) records the standard deviation for the rewards achieved. Note that each population with a single algorithm (`BestPlay, RothErev and Evo`) has a different cloning frequency, while each population with a mix of algorithms has a common frequency (60, 120, 180, 240, 300). This is due to the fact that `Evo` requires at least 20 rounds for speciation to show a difference in behaviour.

Table 9 All mixes of algorithms and cloning frequencies and their associated diversity, gini, reward values

	Cloning frequency	Diversity	Reward	Gini	Std dev
BestPlay	5	1.7	605	2.2	3.830861
	15	1.7	525	7.3	3.140808
	25	1.7	509	18.5	2.813394
	35	1.7	503	23.7	2.711638
	45	1.7	499	24.9	2.802296
	55	1.7	496	25.1	2.857674
	65	1.7	494	24.4	2.970102
	75	1.7	493	22.8	3.053186
	85	1.7	492	21.7	3.142969
	95	1.7	491	19.3	3.249604
	105	1.7	490	18.1	3.321539
BestPlay-Evo	60	2.1	598	0.4	90.388678
	120	2.1	596	0.9	102.229262
	180	2.1	595	1.3	108.460674
	240	2.1	595	1.5	112.134200
	300	2.1	595	1.5	113.784304
Evo	20	2.7	849	−7.3	22.793937
	40	2.7	702	4.3	25.876079
	60	2.7	574	−5.5	35.548009
	80	2.7	516	5.9	32.871732
	100	2.7	504	7.4	26.535574
Evo-RothErev-BestPlay	60	2.6	766	1.0	116.125807
	120	2.5	785	0.9	115.274892
	180	2.5	796	0.8	114.048802
	240	2.5	803	0.7	112.573399
	300	2.5	808	0.8	111.829513
RothErev	5	2.7	931	15.8	2.643791
	15	2.7	952	7.5	5.226056
	25	2.5	953	9.4	7.318583
	35	2.5	950	5.1	10.374762
	45	2.6	948	3.1	13.096147
	55	2.6	946	3.8	19.383790
	65	2.7	946	3.9	22.110698
	75	2.7	946	3.5	28.256625
	85	2.7	946	3.2	31.609064
	95	2.7	945	2.8	36.288345
	105	2.7	945	2.9	37.789350
RothErev-BestPlay	60	2.2	910	1.0	217.470952
	120	2.2	916	1.6	221.771987
	180	2.2	922	1.6	224.047112
	240	2.2	927	2.0	225.288587
	300	2.2	927	2.5	226.053332

Table 9 continued

	Cloning frequency	Diversity	Reward	Gini	Std dev
RothErev-Evo	60	2.7	925	4.0	123.979601
	120	2.7	931	3.4	121.939317
	180	2.7	935	3.0	120.620793
	240	2.7	937	2.7	119.954384
	300	2.7	938	2.7	119.438351

References

1. Baxter, G., & Sommerville, I. (2011). Socio-technical systems: From design methods to systems engineering. *Interacting with Computers*, *23*(1), 4–17. http://iwc.oxfordjournals.org/content/23/1/4.abstract. Accessed 20 Aug 2017.
2. Mohsenian-Rad, A. H., Wong, V. W. S., Jatskevich, J., Schober, R., & Leon-Garcia, A. (2010). Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid. *IEEE Transactions on Smart Grid*, *1*(3), 320–331.
3. Ding, M., Cheng, X., & Xue, G. (2003). Aggregation tree construction in sensor networks. In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, (vol. 4, pp. 2168–2172).
4. Manvi, S. S., Kakkasageri, M. S., & Pitt, J. (2009). Multiagent based information dissemination in vehicular ad hoc networks. *Mobile Information Systems*, *5*(4), 363–389. https://doi.org/10.1155/2009/518042.
5. Yoon, J. H., Baldick, R., & Novoselac, A. (2014). Dynamic demand response controller based on real-time retail price for residential buildings. *IEEE Transactions on Smart Grid*, *5*(1), 121–129.
6. Wu, F., Zsidisin, G., & Ross, A. (2007). Antecedents and outcomes of e-procurement adoption: An integrative model. *IEEE Transactions on Engineering Management*, *54*(3), 576–587.
7. Nallur, V., & Bahsoon, R. (2010). Self-adapting applications based on qa requirements in the cloud using market-based heuristics. In E. Di Nitto & R. Yahyapour (Eds.), *Towards a service-based internet, ser. lecture notes in computer science* (pp. 51–62). Berlin: Springer.
8. Axisa, F., Schmitt, P. M., Gehin, C., Delhomme, G., McAdams, E., & Dittmar, A. (2005). Flexible technologies and smart clothing for citizen medicine, home healthcare, and disease prevention. *IEEE Transactions on Information Technology in Biomedicine*, *9*(3), 325–336.
9. Masoum, A. S., Deilami, S., Moses, P. S., Masoum, M. A. S., & Abu-Siada, A. (2011). Smart load management of plug-in electric vehicles in distribution and residential networks with charging stations for peak shaving and loss minimisation considering voltage regulation. *IET Generation, Transmission Distribution*, *5*(8), 877–888.
10. Emery, F., & Trist, E. (1960). Socio-technical systems. *Management Science, Models and Techniques*, *2*, 83–97.
11. Wooldridge, M. (2001). Intelligent Agents: The Key Concepts. In V. Mařík, O. Štěpánková, H. Krautwurmová, & M. Luck (Eds.), *Multi-agent systems and applications II: 9th ECCAI-ACAI / EASSS 2001, AEMAS 2001, HoloMAS 2001 selected revised papers* (pp. 3–43). Berlin, Heidelberg: Springer.
12. Ladyman, J., Lambert, J., & Wiesner, K. (2013). What is a complex system? *European Journal for Philosophy of Science*, *3*(1), 33–67. https://doi.org/10.1007/s13194-012-0056-8.
13. Pitt, J., Busquets, D., & Riveret, R. (2015). The pursuit of computational justice in open systems. *AI & SOCIETY*, *30*(3), 359–378. https://doi.org/10.1007/s00146-013-0531-6.
14. Chevaleyre, Y., Endriss, U., Lang, J., & Maudet, N. (2007). *A Short Introduction to Computational Social Choice* (pp. 51–69). Berlin: Springer. https://doi.org/10.1007/978-3-540-69507-3_4.
15. Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaitre, J., Maudet, J., et al. (2006). Issues in multiagent resource allocation. *Informatica*, *30*(2), 3–31. https://doi.org/10.1017/S0269888905000470.
16. Balkanski, E., Kurokawa, D., Brânzei, S., & Procaccia, A. D. (2014). Simultaneous cake cutting. In: *Proceedings of the 28th AAAI conference on artificial intelligence*, ser. AAAI'14. AAAI Press, (pp. 566–572). http://dl.acm.org/citation.cfm?id=2893873.2893962. Accessed 20 Aug 2017.
17. Rescher, N. (1966). *Distributive justice: A constructive critique of the utilitarian theory of distribution*. New York: The Bobbs-Merrill Co., Inc.
18. Rescher, N. (2002). Fairness: Theory and Practice of Distributive Justice. In N. Rescher (Ed.), New Brunswick, USA: Transaction Publishers

19. Nallur, V., O'Toole, E., Cardozo, N., & Clarke, S. (2016). Algorithm diversity: A mechanism for distributive justice in a socio-technical MAS. In *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, ser. AAMAS '16. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, (pp. 420–428). http://dl.acm.org/citation.cfm?id=2936924.2936986. Accessed 20 Aug 2017.
20. Odum, E. P. (2004). *Fundamentals of Ecology* (5th ed.). Boston, Massachussets: Cengage Learning.
21. MacArthur, R. (1955). Fluctuations of animal populations and a measure of community stability. *Ecology*, *36*, 533–536.
22. Lawton, J. H., & Brown, V. K. (1994). Redundancy in ecosystems. In E.-D Schulze & H. A. Mooney (Eds.), Biodiversity and ecosystem function (pp. 255–270). Berlin, Heidelberg: Springer.
23. Naeem, S., & Li, S. (1997). Biodiversity enhances ecosystem reliability. *Nature*, *390*, 507–509.
24. Naeem, S. (1998). Species redundancy and ecosystem reliability. *Conservation Biology*, *12*(1), 39–45.
25. Avizienis, A. (1995). The methodology of n-version programming. *Software Fault Tolerance*, *3*, 23–46.
26. Cohen, F. (1993). Operating system protection through program evolution. *Computers and Security*, *12*(6), 565–584.
27. Kc, G. S., Keromytis, A. D., & Prevelakis, V. (2003). Countering code-injection attacks with instruction-set randomization. In: *Proceedings of the 10th ACM conference on computer and communications security*. ACM, (pp. 272–280).
28. Giuffrida, C., Kuijsten, A., & Tanenbaum, A. S. (2012). Enhanced operating system security through efficient and fine-grained address space randomization. In: *Proceedings of the 21st USENIX conference on security symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, (pp. 40–40). http://dl.acm.org/citation.cfm?id=2362793.2362833. Accessed 20 Aug 2017.
29. Snow, K. Z., Monrose, F., Davi, L., Dmitrienko, A., Liebchen, C., & Sadeghi, A.-R. (2013). Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In *Proceedings of the 2013 IEEE symposium on security and privacy*, ser. SP '13. Washington, DC, USA: IEEE Computer Society, (pp. 574–588). https://doi.org/10.1109/SP.2013.45
30. Barrantes, E. G., Ackley, D. H., Palmer, T. S., Stefanovic, D., & Zovi, D. D. (2003). Randomized instruction set emulation to disrupt binary code injection attacks. In *Proceedings of the 10th ACM conference on computer and communications security*, ser. CCS '03. New York, NY, USA: ACM, (pp. 281–289).
31. Shioji, E., Kawakoya, Y., Iwamura, M., & Hariu, T. (2012). Code shredding: Byte-granular randomization of program layout for detecting code-reuse attacks. In *Proceedings of the 28th annual computer security applications conference*, ser. ACSAC '12. New York, NY, USA: ACM, (pp. 309–318). https://doi.org/10.1145/2420950.2420996
32. Davi, L. V., Dmitrienko, A., Nürnberger, S., & Sadeghi, A.-R. (2013). Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and arm. In *Proceedings of the 8th ACM SIGSAC symposium on information, computer and communications security*, ser. ASIA CCS '13. New York, NY, USA: ACM, (pp. 299–310). https://doi.org/10.1145/2484313.2484351
33. Szekeres, L., Payer, M., Wei, T., & Song, D. (2013). Sok: Eternal war in memory. In *Proceedings of the 2013 IEEE symposium on security and privacy*, ser. SP '13. Washington, DC, USA: IEEE Computer Society, (pp. 48–62). http://dx.doi.org/10.1109/SP.2013.13.
34. Lozano, M., Herrera, F., & Cano, J. R. (2008). Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Information Sciences*, *178*(23), 4421–4433. http://www.sciencedirect.com/science/article/pii/S0020025508002867. Accessed 20 Aug 2017.
35. Han, F., & Liu, Q. (2014). A diversity-guided hybrid particle swarm optimization based on gradient search. *Neurocomputing*, *137*, 234–240. Advanced Intelligent Computing Theories and Methodologies-Selected papers from the 2012 18th International conference on intelligent computing (ICIC 2012). http://www.sciencedirect.com/science/article/pii/S0925231214002665. Accessed 20 Aug 2017.
36. Vitorino, L., Ribeiro, S., & Bastos-Filho, C. (2015). A mechanism based on artificial bee colony to generate diversity in particle swarm optimization. *Neurocomputing*, *148*, 39–45. http://www.sciencedirect.com/science/article/pii/S0925231214009412. Accessed 20 Aug 2017.
37. Minku, L., & Yao, X. (2012). Ddd: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, *24*(4), 619–633.
38. Brinker, K. (2003). Incorporating diversity in active learning with support vector machines. In *Proceedings of the 20th international conference on machine learning (ICML-03)* (pp. 59–66).
39. Kuncheva, L. I., & Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, *51*(2), 181–207.
40. Hofmeyr, S. A., & Forrest, S. (1999) Immunity by design: An artificial immune system. In *Proceedings of the 1st annual conference on genetic and evolutionary computation-volume 2*. Morgan Kaufmann Publishers Inc., (pp. 1289–1296).

41. Dasgupta, D. (2006). Advances in artificial immune systems. *IEEE Computational Intelligence Magazine*, *1*(4), 40–49.
42. Pohl, K., Böckle, G., Linden, F. J., & v. d., (2005). *Software product line engineering: Foundations, principles and techniques*. Secaucus: Springer.
43. Wijnstra, J. G. (2000). Supporting diversity with component frameworks as architectural elements. In *Software engineering, proceedings of the 2000 international conference on 2000* (pp. 51–60).
44. Hiltunen, M. A., Schlichting, R. D., Ugarte, C. A., & Wong, G. T. (2000). Survivability through customization and adaptability: The cactus approach. In *DARPA information survivability conference and exposition, 2000. DISCEX '00. Proceedings* (vol. 1, pp. 294–307).
45. Knight, J. C. (2011). Dependable and historic computing. In C. B. Jones & J. L. Lloyd, Eds. Berlin: Springer, ch. Diversity, (pp. 298–312). http://dl.acm.org/citation.cfm?id=2184121.2184147. Accessed 20 Aug 2017.
46. Schaefer, I., Rabiser, R., Clarke, D., Bettini, L., Benavides, D., Botterweck, G., et al. (2012). Software diversity: state of the art and perspectives. *International Journal on Software Tools for Technology Transfer*, *14*(5), 477–495. https://doi.org/10.1007/s10009-012-0253-y.
47. Baudry, B., & Monperrus, M. (2015). The multiple facets of software diversity: Recent developments in year 2000 and beyond. *ACM Computing Surveys (CSUR)*, *48*(1), 16:1–16:26. https://doi.org/10.1145/2807593.
48. Feldmann, R., Gairing, M., Lücking, T., Monien, B., & Rode, M. (2003). *Selfish routing in non-cooperative networks: A survey* (pp. 21–45). Berlin: Springer. https://doi.org/10.1007/978-3-540-45138-9_2
49. Castelli, L., Pesenti, R., & Ranieri, A. (2011). The design of a market mechanism to allocate air traffic flow management slots. *Transportation Research Part C: Emerging Technologies*, *19*(5), 931–943. Freight transportation and logistics (selected papers from ODYSSEUS 2009-the 4th international workshop on freight transportation and logistics). http://www.sciencedirect.com/science/article/pii/S0968090X10001087. Accessed 20 Aug 2017.
50. Sycara, K. P., Roth, S. P., Sadeh, N., & Fox, M. S. (1991). Resource allocation in distributed factory scheduling. *IEEE Expert: Intelligent Systems and Their Applications*, *6*(1), 29–40. https://doi.org/10.1109/64.73815.
51. Kutanoglu, E., & Wu, S. D. (1999). On combinatorial auction and lagrangean relaxation for distributed resource scheduling. *IIE Transactions*, *31*(9), 813–826. https://doi.org/10.1023/A:1007666414678.
52. 't Hoen, P. J., & La Poutré, J. A. (2004). *A Decommitment strategy in a competitive multi-agent transportation setting* (pp. 56–72). Berlin: Springer. http://dx.doi.org/10.1007/978-3-540-25947-3_4
53. Reyes-Moro, A., & Rodríguez-Aguilar, J. A. (2005). *iAuction maker: A decision support tool for mixed bundling* (pp. 202–214). Berlin: Springer. http://dx.doi.org/10.1007/11575726_15
54. Lemaître, M., Verfaillie, G., Jouhaud, F., Lachiver, J.-M., & Bataille, N. (2002). Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, *6*(5), 367–381. http://www.sciencedirect.com/science/article/pii/S1270963802011732. Accessed 20 Aug 2017.
55. Cramton, P., Shoham, Y., & Steinberg, R. (Eds.). (2005). *Combinatorial auctions* (1st ed.). Cambridge: MIT Press.
56. Bredin, J., Maheswaran, R. T., Imer, c., Başar, T., Kotz, D., & Rus, D. (2000). A game-theoretic formulation of multi-agent resource allocation. In *Proceedings of the 4th international conference on autonomous agents*, ser. AGENTS '00 (pp. 349–356). New York, NY, USA: ACM. https://doi.org/10.1145/336595.337525
57. Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, *29*(12), 1104–1113. https://doi.org/10.1109/TC.1980.1675516.
58. Sandholm, T. W., & Lesser, V. R. (2001). Leveled commitment contracts and strategic breach. *Games and Economic Behavior*, *35*, 212–270.
59. Sousa, P., Ramos, C., & Neves, J. (2003). The fabricare scheduling prototype suite: Agent interaction and knowledge base. *Journal of Intelligent Manufacturing*, *14*(5), 441–455. https://doi.org/10.1023/A:1025749208437.
60. Cruz, J. B, Jr., Chen, G., Li, D., & Wang, X. (2004). Particle swarm optimization for resource allocation in uav cooperative control. In A. I. A. A. Guidance (Ed.), *Navigation, and control conference and exhibit*. Providence, USA (pp. 1–11).
61. Yin, P.-Y., & Wang, J.-Y. (2006). A particle swarm optimization approach to the nonlinear resource allocation problem. *Applied mathematics and computation*, *183*(1), 232–242.
62. Tao, F., Zhao, D., Hu, Y., & Zhou, Z. (2008). Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system. *IEEE Transactions on Industrial Informatics*, *4*(4), 315–327.

63. Sen, A. (1970). *Collective choice and social welfare*. Amsterdam: Elsevier.
64. Bentham, J. (1907). *An introduction to the principles of morals and legislation*. Library of economics and liberty, Oxford Clarendon Press, http://www.econlib.org/library/Bentham/bnthPML.html. Accessed 20 Aug 2017.
65. Rawls, J. (1971). *A theory of justice*. Cambridge: Harvard University Press.
66. Pitt, J., Schaumeier, J., Busquets, D., & S. Macbeth, (2012). Self-organising common-pool resource allocation and canons of distributive justice. In *Self-adaptive and self-organizing systems (SASO), 2012 IEEE 6th international conference on* (pp. 119–128).
67. Pitt, J., Busquets, D., & Macbeth, S. (2014). Distributive justice for self-organised common-pool resource management. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *9*(3), 141–1439. https://doi.org/10.1145/2629567.
68. De Jong, K. A. (2006). *Evolutionary computation: A unified approach*. Cambridge: MIT press.
69. McPhee, N. F., & Hopper, N. J. (1999). Analysis of genetic diversity through population history. In *Proceedings of the 1st annual conference on genetic and evolutionary computation*, ser. GECCO'99. (Vol. 2, pp. 1112–1120). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. http://dl.acm.org/citation.cfm?id=2934046.2934071. Accessed 20 Aug 2017.
70. Cobb, H. G., & Grefenstette, J. J. (1993). *Genetic algorithms for tracking changing environments*. DTIC Document: Tech. Rep.
71. Eshelman, L. J., & Schaffer, J. D. (1991). Preventing premature convergence in genetic algorithms by preventing incest. In *ICGA*, (vol. 91, pp. 115–122).
72. Deb, K., & Goldberg, D. E. (1989) An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd international conference on genetic algorithms*. Morgan Kaufmann Publishers Inc., (pp. 42–50).
73. Črepinšek, M., Liu, S.-H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, *45*(3), 1–33.
74. Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, *39*(3), 459–471.
75. Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, (vol. 4, pp. 1942–1948).
76. Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In *Evolutionary computation, 1999. CEC 99. Proceedings of the 1999 congress on*, (vol. 2 pp. 1470–1477).
77. Tereshko, V., & Loengarov, A. (2005). Collective decision making in honey-bee foraging dynamics. *Computing and Information Systems*, *9*(3), 1.
78. Teodorovic, D., & Dell'Orco, M. (2005) Bee colony optimization–a cooperative learning approach to complex transportation problems. *Advanced OR and AI methods in transportation*, (pp. 51–60).
79. Lovbjerg, M., & Krink, T. (2002). Extending particle swarm optimisers with self-organized criticality. In *Evolutionary computation, 2002. CEC'02. Proceedings of the 2002 congress on*, vol. 2, IEEE. Institute of Electrical and Electronics Engineers (IEEE), (pp. 1588–1593).
80. Blackwell, T. M. (2005). Particle swarms and population diversity. *Soft Computing*, *9*(11), 793–802.
81. Kyung-Joong, K., & Sung-Bae, C. (2007). Evolutionary ensemble of diverse artificial neural networks using speciation. Progress in Modeling, theory and application of computational intelligence–15th european symposium on artificial. *Neural Networks*, *71*(7–9), 15.
82. Minku, L. L., White, A. P., & Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, *22*(5), 730–742.
83. Blass, A., Dershowitz, N., & Gurevich, Y. (2008). When are two algorithms the same? *CoRR*, vol. abs/0811.0811, arXiv:0811.0811
84. Challet, D., & Zhang, Y. (1997). Emergence of cooperation and organization in an evolutionary game. *Physica A: Statistical Mechanics and its Applications*, *246*(34), 407–418. http://www.sciencedirect.com/science/article/pii/S0378437197004196. Accessed 20 Aug 2017.
85. Challet, D. (2008). Inter-pattern speculation: Beyond minority, majority and $-games. *Journal of Economic Dynamics and Control*, *32*(1), 85–100. http://www.sciencedirect.com/science/article/pii/S0165188907000413. Accessed 20 Aug 2017.
86. Bianconi, G., Galla, T., Marsili, M., & Pin, P. (2009). Effects of Tobin taxes in minority game markets. *Journal of Economic Behavior & Organization*, *70*(1–2), 231–240. http://www.sciencedirect.com/science/article/pii/S0167268108002084. Accessed 20 Aug 2017.
87. Zapart, C. A. (2009). On entropy, financial markets and minority games. *Physica A: Statistical Mechanics and its Applications*, *388*(7), 1157–1172. http://www.sciencedirect.com/science/article/pii/S0378437108009801. Accessed 20 Aug 2017.

88. Metzler, R., & Horn, C. (2003). Evolutionary minority games: the benefits of imitation. *Physica A: Statistical Mechanics and its Applications*, *329*(3–4), 484–498. http://www.sciencedirect.com/science/article/pii/S0378437103006265. Accessed 20 Aug 2017.
89. Li, Y., & Savit, R. (2004). Toward a theory of local resource competition: the minority game with private information. *Physica A: Statistical Mechanics and its Applications*, *335*(1–2), 217–239. http://www.sciencedirect.com/science/article/pii/S0378437103011348. Accessed 20 Aug 2017.
90. Dhar, D., Sasidevan, V., & Chakrabarti, B. K. (2011). Emergent cooperation amongst competing agents in minority games. *Physica A: Statistical Mechanics and its Applications*, *390*(20), 3477–3485. http://www.sciencedirect.com/science/article/pii/S0378437111003876. Accessed 20 Aug 2017.
91. Greenwood, G. W. (2009). Deceptive strategies for the evolutionary minority game. In *5th International Conference on Computational Intelligence and Games*, (pp. 25–31). http://dl.acm.org/citation.cfm?id=1719293.1719308. Accessed 20 Aug 2017.
92. Mello, B. A., & Cajueiro, D. O. (2008). Minority games, diversity, cooperativity and the concept of intelligence. *Physica A: Statistical Mechanics and its Applications*, *387*(2–3), 557–566. http://www.sciencedirect.com/science/article/pii/S0378437107009995. Accessed 20 Aug 2017.
93. Galla, T., Mosetti, G., Zhang, Y.-C. Anomalous fluctuations in minority games and related multi-agent models of financial markets.
94. Ranadheera, S., Maghsudi, S., & Hossain, E. (2017). Minority games with applications to distributed decision making and control in wireless networks. *IEEE Wireless Communications*, (vol. PP, no. 99, pp. 2–10).
95. Zhang, C., Wu, W., Huang, H., & Yu, H. (2012). Fair energy resource allocation by minority game algorithm for smart buildings. In: *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (*2012*, pp. 63–68).
96. Shannon, C. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*, 379–423.
97. Ashlock, D. (2006). *Evolutionary computation for modeling and optimization*. New York: Springer.
98. De Jong, K. (2006). *Evolutionary computation: A unified approach*. Cambridge: MIT Press.
99. Eiben, A. E., & Smith, J. (2015). From evolutionary computation to the evolution of things. *Nature*, *521*, 476–482.
100. Roth, A. E., & Erev, I. (1995). Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, (*8*(1), pp. 164–212). http://linkinghub.elsevier.com/retrieve/pii/S089982560580020X. Accessed 20 Aug 2017.
101. Nallur, V., Cardozo, N., & Clarke, S. (2016). Clonal plasticity: A method for decentralized adaptation in multi-agent systems. In *Proceedings of the 11th international workshop on software engineering for adaptive and self-managing systems*, ser. SEAMS '16. New York, NY, USA: ACM, (pp. 122–128). https://doi.org/10.1145/2897053.2897067
102. Harper, J. L. (1980). Plant demography and ecological theory. *Oikos*, *35*(2), 244–253.
103. Gini, C. (1936). On the measure of concentration with special reference to income and statistics. *Colorado College Publication*, *208*, 73–79.
104. Lehman, J., & Stanley, K. O. (2011). Improving evolvability through novelty search and self-adaptation. In *2011 IEEE congress of evolutionary computation (CEC)*. Institute of Electrical and Electronics Engineers (IEEE).
105. Beal, J. (2011). Functional blueprints: An approach to modularity in grown systems. *Swarm Intelligence*, *5*(3), 257–281.
106. Cully, A., Clune, J., Tarapore, D., & Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, *521*, 503–507.
107. Smith, D., Tokarchuk, L., & Wiggins, G. (2016). Rapid phenotypic landscape exploration through hierarchical spatial partitioning. In *International conference on parallel problem solving from nature*. Springer, (pp. 911–920).
108. Pugh, J. K., Soros, L. B., & Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, *3*, 40.
109. Sareni, B., & Krähenbühl, L. (1998). Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, *2*(3), 97–106.
110. Mouret, J. B., & Doncieux, S. (2012). Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, *20*(1), 91–133. https://doi.org/10.1162/EVCO_a_00048.
111. Hamann, H. (2015). Lessons from speciation dynamics: How to generate selective pressure towards diversity. *Artificial Life*, (*21*(4), pp. 464–480). http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=110916959&site=ehost-live. Accessed 20 Aug 2017.

112. Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International conference on parallel problem solving from nature*. Springer, (pp. 849–858).
113. Stillger, M., Spiliopoulou, M. (1996). Genetic programming in database query optimization. In *Proceedings of the 1st annual conference on genetic programming*. Cambridge, MA, USA: MIT Press, (pp. 388–393). http://dl.acm.org/citation.cfm?id=1595536.1595591. Accessed 20 Aug 2017.
114. Augusto, D. A., & Barbosa, H. J. C. (2000). Symbolic regression via genetic programming. In *Proceedings Sixth Brazilian Symposium on Neural Networks* (Vol. 1, pp 173–178).
115. Bazzan, A. L. C., Bordini, R. H., Andrioti, G. K., Vicari, R. M., & Wahle, J. (2000). "Wayward agents in a commuting scenario (personalities in the minority game)," In *Proceedings Fourth International Conference on MultiAgent Systems*, pp. 55–62.