Auton Agent Multi-Agent Syst manuscript No.
(will be inserted by the editor)

On environment difficulty and discriminating power

José Hernández-Orallo

March 26, 2014

Abstract This paper presents a way to estimate the difficulty and discriminating power of any task instance. We focus on a very general setting for tasks: interactive (possibly multi-agent) environments where an agent acts upon observations and rewards. Instead of analysing the complexity of the environment, the state space or the actions that are performed by the agent, we analyse the performance of a population of agent policies against the task, leading to a distribution that is examined in terms of *policy* complexity. This distribution is then sliced by the algorithmic complexity of the policy and analysed through several diagrams and indicators. The notion of environment response curve is also introduced, by inverting the performance results into an ability scale. We apply all these concepts, diagrams and indicators to two illustrative problems: a class of agent-populated elementary cellular automata, showing how the difficulty and discriminating power may vary for several environments, and a multi-agent system, where agents can become predators or preys, and may need to coordinate. Finally, we discuss how these tools can be applied to characterise (interactive) tasks and (multi-agent) environments. These characterisations can then be used to get more insight about agent performance and to facilitate the development of adaptive tests for the evaluation of agent abilities.

Keywords Environment difficulty · agent evaluation · discriminating power · agent policy · algorithmic information theory · universal psychometrics · reinforcement learning · elementary cellular automata.

1 Introduction

Determining the difficulty and the discriminating power of a task or problem instance is a fundamental issue in many disciplines and applications. It is especially necessary for the evaluation of natural and artificial systems, as usually done in psychometrics, animal cognition and artificial intelligence [36]. The capabilities of a system, agent or technique are given by the kinds and difficulty of the tasks it can solve. Tests (mostly in comparative psychology and psychometrics) and competitions (mostly in artificial intelligence [59, 1, 24]) are usually designed to cover some range of capabilities and difficulties, and efforts are made to include tasks that are most informative. In this sense, tasks become discriminating if they can be solved by some agents (or techniques) but not by others. As a result, it is of utmost importance to precisely determine not only the types of tasks that are used to build tests and benchmarks but also their difficulty and their discriminating power. Unfortunately, the measures of difficulty and discriminating power in artificial intelligence are usually estimated in an informal way, blurred or not considered at all. In fact, it is not always well understood that difficulty is associated with the expected result for a set of agents while discriminating power is associated with the variability of the results for a set of agents.

The analysis of difficulty and discriminating power has been a recurrent issue in psychometrics. Item Response Theory (IRT) [19], for instance, is a well-founded approach in psychometrics where the difficulty (in terms of response curves) for each item is used to construct more effective tests, in order to derive

DSIC, Universitat Politècnica de València, Spain, E-mail: jorallo@dsic.upv.es

scores and to obtain reliability measures. The "difficulty" of an item or task is generally (but not always) calculated with the results of previous tests on the same population, and not as a result of an underlying theory of the intrinsic problem difficulty of the task at hand. In artificial intelligence, the emphasis has not usually been put on evaluation, but on the development of intelligent systems. Nonetheless, some domain-specific tests and benchmarks have usually been designed by including problem instances of various difficulty levels. In the end, the use of tasks of previously known difficulty and discriminating power is crucial to make testing effective, and get an accurate result with a minimum amount of tasks and time.

In more general terms, the concepts of complexity and difficulty appear in almost every field of science: physics, biology, psychology, (algorithmic) information theory, complex system theory, evolutionary computation, cryptography and many other fields. The analyses and measures are so diverse that any comprehensive account is far beyond any survey, although we will give some pointers in the next section. The reason of such a diversity is due to the very different elements whose complexity we may be interested in (physical phenomena, life forms, bit strings, algorithms, reasoning processes, etc.) and also the different purposes (understanding phenomena, developing new methods or just measuring individuals). In fact, it is when we talk about agents (or any other kind of cognitive system) that we use the word 'difficulty' instead of 'complexity', as suggesting that 'difficulty' is a relative (cognitive) issue. In other words, complexity is usually associated with the problem (statement and solution) and difficulty is usually associated with the way or method to go from statement to solution.

In this paper, we focus on the analysis of difficulty (and discriminating power) for a particular (but ultimately general[1]) setting. We concentrate on an interactive scenario, where an agent interacts with an environment through actions and observations. We consider (possibly infinite) tasks where performance is evaluated in terms of rewards (or a utility function, which can represent the degree of achievement of a goal). This is in line with many approaches for cognitive system evaluation in psychology, animal cognition and artificial intelligence, most especially in the reinforcement learning setting, in game theory and in multi-agent systems. In the context of agent-environment interaction, the task is represented by the environment and the good 'solutions' are given by sequences of actions that maximise rewards. Several studies in artificial life and multi-agent systems have focussed on the way in which small changes can affect the behaviour of the system in terms of cycles, deadlocks, etc. (see, e.g., [20]). However, this analysis has not usually been extended to the sensitivity and influence on agent performance (in terms of the probability of goal achievement or expected reward) after minor changes on the environment (or the inclusion of other agents). This sensitivity on performance results is closely related to the problem of difficulty we want to address here.

Apart from rewards, we focus on a second dimension: policy complexity. Actions are the result of a policy, which determines a behaviour in general. The central idea of this paper is to put the emphasis on behaviours, rather than actions. So taking these two dimensions into account (rewards and policies), we highlight the issue that the aggregated rewards for the possibly infinite set of policies may be distributed in many different ways. As we will see, the study of these distributions plays a crucial role in the estimation of task difficulty. Another distinctive feature of our approach is that we will assess environment difficulty using algorithmic information theory [50] (by estimating the Kolmogorov complexity of each policy) in order to (1) evaluate the information content of the policy, (2) make the complexity measure more independent of the representation language, due to the invariance theorem[2], and (3) consider all (or many of) the alternative (but equivalent) expressions of the same policy, so making the estimation more robust, as with Solomonoff's algorithmic probability (this is exploited by the coding theorem method [14,64]).

So, the main idea of this paper is that we look at environment difficulty from the standpoint of the agent policies, by calculating their complexity. We do not estimate the complexity of the environment (including or not the rest of agents, when it is a multi-agent system). We do not estimate the complexity of the solution either (the sequence of actions). Instead, we estimate the complexity of the policies, using a policy description language (an agent language) and compressing each program. As a result, we evaluate

[1] Many other problems can be formulated using restricted environment classes (see, e.g., the hierarchy of environments in [45, ch. 3]).

[2] The Kolmogorov complexity of the same object using two different description mechanisms is the same, up to a constant that is independent of the object [50].

the difficulty of an environment by observing the distribution of policies in terms of their Kolmogorov complexity and their aggregated reward. The most straightforward analysis of the distribution may just look at the frequency of simple policies achieving good results. In other words, we could start with the following simple 'maxim': "*If the environment produces good results for many simple policies then it is easy. Otherwise, it is difficult*". After this starting point, we will also derive other more robust indicators, functions and graphical tools from this distribution, in order to say when an environment is more or less difficult. Also, we will investigate whether the environment is discriminating, not only in the beginning but after a random walk or after other policies. This notion is crucial in the context of evaluation, as an environment or task for which all policies of a given complexity scale fail is useless to discriminate between different methods around that scale.

We understand that problems are addressed by policy-searching *methods* that try many policies. This interpretation is crucial, as AI systems do not restrict themselves to just one policy, especially when we refer to learning agents or systems that explore many policies (such as evolutionary techniques). Some policies may be more successful than others, and some may be easier to find, explore or implement than others. Difficulty and discriminating power are derived from these two concepts: the expected policy quality, which will be formalised as expected reward, and policy weights, which will be formalised as a complexity and policy distribution. This leads to the study of this landscape of policies where policy-searching methods struggle. So the goal of this paper is to develop new theoretical and graphical tools to analyse the difficulty and discriminating power of a given problem based on a population of solving policies, which are developed from a common language and analysed in terms of their distribution and complexity. While the approach is illustrated in the context of interactive systems using rewards, the techniques are applicable to the analysis or development of AI systems and multi-agent systems. In particular, environments can be characterised by their *response curves* and so arranged as a battery of tasks that can be used to perform more efficient and accurate evaluation protocols, benchmarks and competitions through adaptive testing.

The rest of this paper is distributed as follows. Section 2 shortly reviews some of the many different ways of measuring complexity that have appeared in the literature, distinguishing those that look at the problem (statement or solution), at the search space, at the population of solvers or other competing agents. Also, several notions of difficulty and discriminating power are introduced, as well as the basic definition of multi-agent system we will work with throughout the paper. Section 3 analyses the whole distribution of policy complexities and aggregated rewards, using graphical tools, summary indicators and adapting the notion of response function, which finally leads to some general indicators of difficulty and discriminating power. Section 4 introduces a case study with a minimalistic environment class based on *agent-populated elementary cellular automata* and a very simple agent policy language. Using them, we show the distribution of rewards according to policy complexity, calculate their indicators and plot their response functions. Section 5 illustrates the applicability of distribution plots and response functions with a different multi-agent environment, which includes some cooperative and competitive features, and is analysed in terms of homogeneous and heterogeneous collectives and in terms of competing individuals. Section 6 discusses how this approach relates to other kinds of complexity and develops the issue of more abstract policy languages and their limitations. As a take-home message, section 7 distils a step-by-step procedure that describes how the approach that is presented in this paper can be (systematically) applied to the evaluation of environments. Finally, section 8 analyses the general applicability for the assessment of artificial agents in (multi-agent) environments and other areas of AI and explores avenues for future work. The appendix includes some proofs.

2 Background

There is a plethora of points of view around the notion of 'problem difficulty' and related terms such as 'task complexity' or 'instance hardness'. We will shortly review some of them. Instead of arranging them according to their origin: psychometrics, computer science, complexity theory, biology, etc., we will group the approaches in the literature according to what they look at in order to see how they establish measures of difficulty or discriminating power.

2.1 Looking at the problem

The first distinction that we need to do is between problem *class* difficulty and problem *instance* difficulty. In computational complexity theory [17], the focus is usually put on class complexity, e.g., the behaviour of an algorithm for all possible inputs, in terms of the time (or space) that an algorithm takes to solve a problem or to decide whether an object belongs to a set. Nonetheless, the notion of instance complexity has also been considered, related to the concept of average-case computational complexity, developed by Leonid Levin in the 1980s [49], which is of course related to the more general notion of average-case performance of algorithms (see, e.g., [43]). However, the point of view is the time (or space) that a *given* algorithm (or the best possible algorithm[3]) requires to solve a problem instance, such as quicksort being faster for an almost sorted instance than a randomly sorted one. Hence, it is not about the *intrinsic* complexity of the problem instance, e.g., sorting "acfgdbe". One of the most insightful approaches is the analysis of the SAT problem (and variants), since they cover (by reduction) a wide range of algorithmic problems, and the features that make a problem difficult or not have been explored in the past decades [25,40], but many questions still remain.

Here we are interested in instance *difficulty*, from the point of a view of cognitive agent evaluation (running whatever algorithm or policy). This stance has been taken by psychology and other cognitive sciences, where there is an old history of approaches using the concept of *working memory* or the number of elements that must be considered at the same time in order to solve a problem [55,4,28,6]. Note that this is about the complexity of figuring out the solution in terms of other components the subject already knows, but needs to combine. In this line, and closely related to the emerging field of artificial intelligence, Simon and Kotovsky explored, for decades, "the problem space of difficulty" [58,44]. However, most of these views of difficulty have been anthropocentric. The use of these measures for non-human subjects (especially for machines) may be misleading since it has been shown that many tasks that are very difficult for humans are very easy for state-of-the-art machines and vice versa [16].

A more mathematical (and computational) approach for associating complexity with the number (or the size) of items that are necessary to explain a concept (or solve a problem) is now known as algorithmic information theory (also known as Kolmogorov complexity [50]). As a result, in the past forty years, the difficulty of solving a particular task has been occasionally related to its Kolmogorov complexity. This relation has been especially advocated for in inductive inference, because it makes sense to look at the length of the shortest pattern explaining the evidence, and this length seems to determine the difficulty of the problem. However, the relation is, at most, unidirectional[4], and not always intuitive[5]. Several alternatives, such as Levin's Kt [48], logical depth [10], effective complexity [5], computational depth [3], sophistication [11], and others (see [50, chap. 7]), have been proposed instead, where not only the solution of the problem (the pattern) is analysed but also how difficult it is to extract that solution from the evidence. These proposals are generally conceived for the evaluation of (static) objects but some of them can also be used for quantifying difficulty in sequential inductive problems. For instance, a variant of Kt, known as intensional complexity, accurately captured the difficulty humans found on IQ test series problems [38,30]. Outside induction, the idea of instance complexity [56][50, sec. 7.4] has also been developed with the use of algorithmic information theory, in front of the classical view of problem class complexity mentioned above. For other kinds of problems, such as general inductive and deductive problems, [32,31] also explored the use of algorithmic information theory, by considering the information gain from the problem to the solution.

The above approaches have mostly focussed on static or (at most) sequential problems. When we move to the evaluation of natural or artificial agents, we need to address the estimation of the difficulty of *interactive* tasks, or *environments*, where subsequent input is affected by the agent's actions. Here, we usually consider other interactive elements as well, other than actions and observations, such as a score or a reward. In reinforcement learning, for instance, the difficulty of a problem depends on the goal but also on the way that rewards are produced (continuously or not, and according or not to the ultimate

[3] We may consider average cases or problems, but not an average algorithm (which is not the same as a random algorithm).

[4] Some repetitive, but long, patterns (hence having high Kolmogorov complexity) may just require memory to see and store the repetition.

[5] True random strings are incompressible, so leading to no 'solution' at all, either easy or difficult.

goal). The evaluation of interactive environments is more challenging. For instance, each action leads to a different (sub)environment, so the results may be very sensitive to an early calamitous or auspicious action, as in the heaven-or-hell environment [60, sec. 3.2]. This is why some works (in the context of evaluation) have advocated for ergodic environments [45], where the agent can always recover from a local 'hell' or 'heaven'. Yet another issue is that estimating the reward of a policy may take many steps (and repetitions) in the environment. Difficulty and, most especially, discriminating power are strongly affected by the issue of environment responsiveness. Some environments can be unresponsive from the very start. This makes them very 'difficult' (if they start in a 'hell' state), but of low interest as they do not have any discriminating power. In the end, environment unresponsiveness can be caused by many different reasons, such as the inability or impossibility of increasing rewards (even though the agent can still do a variety of actions), the impossibility of taking actions (a trap) because a collision against some system rules or the collision against the actions of other agents, just because the environment has become chaotic due to a highly random behaviour of the system, or, in multi-agent systems, when environments become too crowded or full of random agents. The existence of so many causes for unresponsiveness suggests that a better way to address this issue may be in terms of discriminating power, with a more global view of how policies affect rewards.

Despite all these problems, many works on reinforcement learning, agents and robotics have considered specific ways of approximating difficulty. Typically, the mere 'size' of the problem (e.g., maze size, [62]) or the state space [52], have been used as approximations of the 'difficulty' of an environment. However, these approaches are usually domain-specific and are prone to a series of problems because difficulty is just handled in an informal or ad-hoc way.

Alternatively, the application of Kolmogorov complexity to environments has also been explored [46,45]. Although not explicitly used as an estimation of difficulty (but rather as a way to derive a distribution), this approach becomes much more cumbersome [39] than it might seem, because the relation between complexity and difficulty is intricate. In [34], another variant of Kolmogorov complexity ($Kt^{\max}$) was suggested as a measure of complexity for the environments, but it was already stated that this approximation was unidirectional: some very complex environments might be easy, i.e., high rewards could be obtained by very simple policies ([34, sec. 4.1]). Also, the problem of whether some environments are more *discriminating* than others was also discussed there, and some notions (such as reward-sensitiveness) were introduced [34, sec. 4.2]. Following these ideas, an environment class was introduced in [33], where the difficulty of an environment can be approximated by the Kolmogorov complexity of the rules (expressed with a Markov algorithm) that describe the environment. The class was used to create intelligence tests in [42], but the approximation of difficulty was not satisfactory.

A more minimalistic approach, starting with very basic components is the study of the emergence of complexity (and other properties) in cellular automata and other artificial life universes (see, e.g., [61]). Algorithmic information theory has also been applied here; an interesting approach is the calculation of the complexity of the patterns that an elementary (1-dimensional) cellular automaton generates [65]. However, the purpose of Zenil et al. is not to measure the difficulty for an agent (not even a 'glider'), but whether the 2-dimensional patterns have low or high complexity. In fact, it is very unlikely (or only likely after billions of generations) that in any of these settings or any other artificial life universe, we could find 'agents' with some non-trivial cognitive abilities (see, e.g., [15, sec. 0.2.7, p545, col. 1]).

2.2 Looking at the search space

Instead of looking at the complexity of the environment, a natural way to assess the difficulty of a problem relies on analysing the search space. For instance, in a maze, the search space is the set of trajectories that can be performed in the maze. Each trajectory is a sequence of actions. Analysing how likely the solutions are in the space of trajectories may work because, in this particular case, the environment is static (the maze does not change after the agent's actions). However, things are different for other kinds of environments, if there is a reaction to the agent's actions, or, most especially, in multi-agent systems. Nonetheless, with an appropriate abstraction of the a search space, the same ideas and principles used in the areas of optimisation and heuristics can be applied here. For instance, it is said that a problem is difficult if there is a "high density of well-separated almost solutions (local minima)" [12] or we have a "rugged solution space" [40]. This view of difficulty is common in constraint satisfaction problems, machine learning, tree search algorithms, simulated annealing, etc.

Evolutionary computation is an area where the search space and its relation to difficulty has been studied more exhaustively. In evolutionary computation we consider the notion of operator, which converts (mutates) one solution into a different solution. The solutions are the nodes of the search space and the operators are edges. This makes up the 'landscape' in evolutionary approaches. However, it is not clear how to adapt this to other settings. For instance, in many evolutionary approaches the size of the individual codes (genotype) is constant. Only when the genotype size is variable (e.g., evolutionary programming), the size of the solution is variable. But, even in this case, the size is not clearly related to difficulty. He et al. [29] include an account of approaches for problem difficulty measures in evolutionary computation. There are several interesting concepts such as "rugged fitness landscape" (problems where the solution space is full of local minima and maxima), the 'needle in a haystack" metaphor, the existence of one (or a few) way to the solution, *deceptive* ways (promising ways finally leading nowhere), the appearance of "epistatic interactions" between parts of the solution (gene co-influence), the notions of building blocks, and many others.

2.3 Looking at the population of solvers

Apart from the problem, solution and search space, we can focus on the solver. This is the approach taken in psychometrics to assess problem (item) difficulty, where item difficulty is derived by observing previous performance of a population of subjects on that item. From this perspective, items are usually classified into several difficulty categories, and a variety of items of different difficulty is used in tests in order to cover a wide range of the ability that is to be measured. In this approach, subjects are just evaluated about their behaviour, and it is irrelevant here how each subject works internally or the (computational) complexity of its inner processing.

Item response theory (IRT) [19] is a paradigm for the study of items (tasks) and a well-grounded way of designing tests and other instruments that measure abilities, especially in the area of (computerised) adaptive testing. IRT estimates mathematical models to infer the associated probability and informativeness estimations for each item. One very common model for discrete-score problems is the three-parameter logistic model, where the item response function (or curve) corresponds to the probability that an agent with ability θ gives a correct response to an item. This model is characterised as follows:

$$p(\theta) \triangleq c + \frac{1-c}{1+e^{-a(\theta-b)}}$$

where a is the *discrimination* (the maximum slope of the curve), b is the *difficulty* or item location (the value of θ leading to a probability half-way between c and 1, i.e., $(1+c)/2$), and c is the chance or asymptotic minimum (the value that is obtained by *random* guess, as in multiple choice items). The zero-ability expected result is given when $\theta = 0$, which is exactly $z = c + \frac{1-c}{1+e^{ab}}$. Figure 1 (left) shows an example of a logistic item response curve.

For continuous score items, a very frequent approach is the linear model [53,22]:

$$X(\theta) \triangleq z + \lambda\theta + \epsilon$$

where z is the intercept (zero-ability expected result), λ is the loading or slope, and ϵ is the measurement error. Again, the slope λ is positively related to most measures of discriminating power [23]. Figure 1 (right) shows an example of a linear item response curve. Note that for continuous score items, if they are bounded, the logistic model may be more appropriate.

Working with item response models is very useful for the design of tests, because if we have a collection of items, we can choose the most suited one for the subject (or population) we want to evaluate. According to the results that the subject has obtained on previous items, we may choose more difficult items if the subject has succeeded on the easy ones, we may look for those items that are more discriminating (i.e., more informative) in the area we have doubts, etc. Note that discrimination is not a global issue: a curve may have a very high slope at a given point, so it is highly discriminating in this area, but the curve will

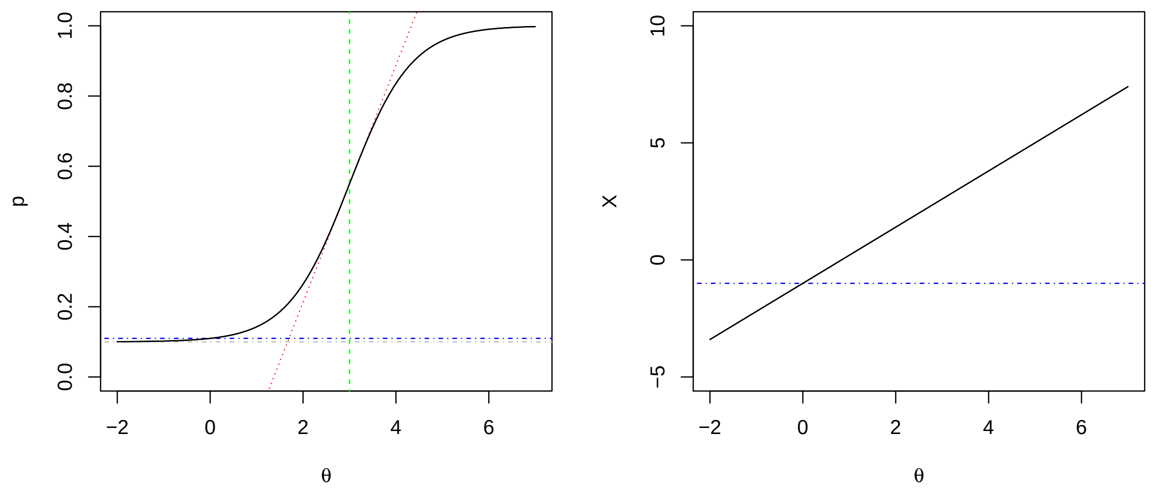


Fig. 1 Left: item response function (or curve) for a binary score item with the following parameters for the logistic model: discrimination $a = 1.5$, item location $b = 3$, and chance $c = 0.1$. The discrimination is shown by the slope of the curve at the midpoint: $a(1 - c)/4$ (in dotted red), the location is given by b (in dashed green) and the chance is given by the horizontal line at c (in dashed-dotted grey), which is very close to the zero-ability expected result $p(\theta) = z$ (here at 0.11). Right: A linear model for a continuous score item with parameter $z = -1$ and $\lambda = 1.2$. The dashed-dotted line shows the zero-ability expected result.

almost be flat when we are far from this point. Conversely, if we have a low slope, then the item covers a wide range of difficulties but the result of the item will not be so informative as for a higher slope.

One important question about these curves is how the parameters are estimated, i.e., how the difficulty and discriminating power are estimated. This is usually done by applying the item to a *population* of subjects for which we already know their ability. Since the ability of subjects is usually determined by tests, this leads to a certain circularity, which is usually sorted out by psychometricians with an incremental approach: the first tests are devised by comparing on informal assessments of both the ability of subjects and the difficulty of tasks, the next attempt refines them using the results from the previous ones, etc. It is important to note that there is no theoretical assessment of the intrinsic difficulty or discriminating power of items: this is done empirically by looking at the population. In other words, difficulty is assessed relative to a population, which can be fixed in terms of biological or social grounds, as the population of adult humans is used to normalise IQ results on a scale.

With *natural* populations (e.g., adult humans, children, chimpanzees, etc.) it is meaningful (and feasible) to make a sample. However, in this paper, we face the case of 'artificial populations', i.e., distributions of agents. This is more cumbersome, and we may take two different approaches. One option is to select a family of subjects from the set of systems that have been devised in AI. For instance, we could make a selection of reinforcement learning systems as a population to evaluate the complexity of a specific maze problem. This approach is frail, as the selection of the population will always be arbitrary (with some ad hoc specialised systems being or not selected affecting the measures dramatically) and determined by current technology, making our difficulty estimations being subjective to the state of the art in AI. A second option is to define a language of expressing solutions to the problem (or policies) and define the population as the set (or distribution) of solutions (or policies) that can be defined using that language. This is the approach that we will take in this paper.

2.4 Looking at other (competing) agents

We have seen how challenging it is to define an objective view of difficulty for a given problem, when there is only one agent which is evaluated against the task or environment. Even more challenging is when we consider other agents in the environment. In games (and multi-agent systems in general), the difficulty of a problem depends on the rules, but most especially on the opponents. For general multi-agent systems and multi-agent reinforcement learning [9] in particular (and their competitions [24]), the difficulty of the system depends on the opponents and cooperators [41], and their intelligence [35,37]. In fact, some environments may have a rich, sensitive behaviour with some agents, but can have a very different, chaotic (or unresponsive) behaviour with other agents. Apart from some particular cases, such as those studied in game theory, it is virtually impossible to determine in general whether these (good or bad) equilibria exist and how they depend on the number and kind of agents, as common in multi-agent systems, artificial life, natural ecology, etc. This means that we need to evaluate case by case.

Technically, we can consider the other agents as part of the environment, and treat everything as a whole. However, this is not a very practical approach, as we usually do not have a complete specification of the other agents or a full account of their behaviour. This makes the approaches in sections 2.1 and 2.2 even more unrealistic in this case. The population approach (as seen in section 2.3) seems more appropriate, but has not been fully explored as a systematic and scientific measuring approach. For instance, in psychometrics, social abilities are not usually evaluated by confronting subjects in social contexts with other subjects. Nonetheless, this is actually the usual approach in games and sports. For instance, in chess, the difficulty of a professional match is given by the Elo rating [18] of the opponent. While this kind of rating can be extended to games and multi-agent systems with more than two opponents, and even in cases where individuals play for teams, this approach still depends on the population that is chosen.

Finally, the concept of discriminating power in a multi-agent system (or a multi-player game) must be associated to the notion of stability or predictability. There are games where the result against a specific player with a given rating is highly predictable (e.g., playing against the best player) while other games show more variability (due to random effects or due to the rules of the game). The expected value an be approximated when several matches are held. In any case, it is usually more informative to play against a mid-rated player or, instead, to adjust the opponent to the given rating of the subject that we want to evaluate. This is the way many tournaments (e.g., chess or tennis) are designed. All this gives support as well to an approach based on discriminating power (variability of results for different policies) instead of more obscure or restricted concepts such as unresponsiveness, sensitiveness, chaoticness or ergodicity.

2.5 Multi-agent environments, actions and policies

The main goal of this paper is to study the difficulty that is posed by a given environment for an interacting system, an agent, which explores many policies in order to solve the problem (in terms of achieving a goal or improving a utility function). In order to work with a very general setting (including games, for instance), we will consider that there might be *other* agents in the environment. That means that we need to distinguish between the evaluated agent and the other agents that become part of the environment. We will use the term 'role' for this.

There are many possible definitions of multi-agent environments. Usually, the definitions are based on some common space, transition rules and reward system (or goals), which are shared by all of them. A possible, more formal definition follows:

Definition 1 A multi-agent environment μ is a tuple $\langle \mathcal{S}, \tau, \omega, \rho, \Pi \rangle$ where $\mathcal{S}$ is a (state) space, τ is a state transition function or probability function, Π is a set of agents $\pi_1, \pi_2, \ldots, \pi_n$ (policy functions) acting on roles $1, 2, \ldots, n$, and ω and ρ are observation and reward functions (respectively) for all the agents.

The previous definition does not completely specify the topology of the space, the possible contents, the transition rules, the reward system and the movements of the agents. We will see specific cases in sections 4 and 5. For the moment, it is enough to analyse the issue of difficulty in a general, but still formal way, with the following definitions and notation. Given any of the agents π_i running for t steps on a environment, we denote by χ_i^t its interaction history, which is a sequence of tuples of the form $\langle a, o, r \rangle$, representing action, observation and reward respectively, and where a and o are elements in finite discrete sets $\mathcal{A}$ and $\mathcal{O}$ respectively, and r is a bounded rational number. The policy π_i is then a function $\pi_i(\chi_i^t) \to a_i^{t+1}$. Similarly, the transition function takes the previous state and the actions of all the agents and returns the new state: $\tau(\sigma^t, a_1^t, a_2^t, \ldots, a_n^t) \to \sigma^{t+1}$. If the environment is stochastic, then $\tau(\sigma^{t+1} | \sigma^t, a_1^t, a_2^t, \ldots, a_n^t)$ is a conditional probability distribution. Once the state is updated, observations and rewards are produced for all the agents: $\omega(\sigma^{t+1}, i) = o_i^{t+1}$ and $\rho(\sigma^{t+1}, i) = r_i^{t+1}$.

We have a simple transition function that takes all actions into account and gives a new state. This does not mean that any attempted action by an agent is accomplished, as this transition function may include (and apply) constraints about conflicting actions with some environment rules (such as moving to a forbidden cell) or with other agents. Other multi-agent system models prefer to distinguish between influences and reactions (such as Ferber and Müller's model of influence and reaction [21], or some further evolutions of the model, such as [54]). In our second case study in section 5 we will see how the actions of several agents are considered simultaneously. Some filters are applied in order to see which agents can perform their action and which cannot (because they collide on the same cell).

We now need a few definitions. We denote by $R^t_i(\pi, \mu)$ an aggregation function of the rewards that are obtained by π in role i of environment μ until time t (note that the agent in role i, i.e., π_i in $\varPi$, is instantiated with π). For stochastic environments, this represents an expected value. Given a class of policies or agents $\varOmega$, we define:

$$R_{\max}{}^t_i(\mu) \triangleq \max_{\pi \in \varOmega} R^t_i(\pi, \mu) \tag{1}$$

$$R_{\min}{}^t_i(\mu) \triangleq \min_{\pi \in \varOmega} R^t_i(\pi, \mu) \tag{2}$$

$$R_{\text{mean}}{}^t_i(\mu) \triangleq \frac{1}{|\varOmega|} \sum_{\pi \in \varOmega} R^t_i(\pi, \mu) \tag{3}$$

as the maximum, minimum, and average (respectively) aggregated (expected, if the environment is stochastic) reward attainable in t steps. In other words, this is the best, worst, or average (respectively) score for any possible agent in $\varOmega$ performing with role i in the environment.

From now on, we may drop i when it is clear which role we are using (or there is only one agent in the environment, i.e., if it is not a multi-agent environment) and we may drop t when clear from the context (or irrelevant for the matter in hand). Similarly, we will also drop μ when working with just one environment.

3 Agent policy distribution

The complexity of the problem formulation (the environment) or the solution (the sequence of actions), do not provide good ways for deriving a measure of environment complexity. In the case of environments, some may have a very complex description, but may be solved easily. If we focus on the sequence of actions, we have even more problems. First, the sequence of actions depends on the environment. For instance, consider a very complex environment which outputs a sequence of non-compressible (i.e., complex) observations and consider that the best reward is just attained by performing an action which is a simple function of the observation (e.g., if observations and actions were binary, just replicating the input as an output). Then the complexity of the sequence of actions would be high, but the policy the problem requires would be intuitively easy. Second, action sequences depend on t and may become very long for long episodes. In contrast, policies are finite, as they are programs that represent agents, i.e., behaviours, not sequences of actions. In the end, we are interested in knowing whether there are simple agents (i.e., simple policies) solving the problem. Finally, focussing on agent policies rather than sequence actions makes it much easier to deal with stochastic environments.

Consequently, we are going to consider agent policies instead. We can just calculate the complexity of the best policy or, more precisely, the lowest complexity of any best policy, known as the best-policy difficulty (or hardness) $\hbar$ as follows:

$$\hbar^t_i(\mu) \triangleq \min_{\pi : R^t_i(\pi,\mu) = R_{\max}{}^t_i(\mu)} K(\pi) \tag{4}$$

Unless precisely specified, K can be the Kolmogorov complexity, an approximation or other related function. Intuitively, if the best policy (or one of the best policies) has low complexity, the problem could be said to be easy. While this is a straightforward interpretation, it is too naive. It might be the case that the best policy π leads to $R(\pi) = r$ while the second best policy π' leads to $R(\pi') = r - \epsilon$. If ϵ is (comparatively) very small but $K(\pi') \ll K(\pi)$, we may even say that an almost equally good (but much

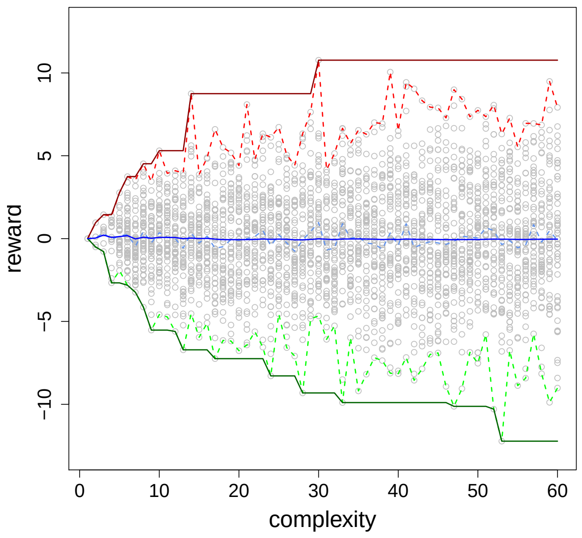


Fig. 2 Figurative plot illustrating the difficulty of a balanced environment (an environment where random actions lead to an expected reward of 0, as in [34,33, sec. 4.2]). It shows a distribution of the expected aggregated reward R (y-axis) for an arbitrarily large number of steps, according to the agent's complexity k (x-axis). It does not show all the possible agents π with $K(\pi) \leq 60$ but a sample containing about 50 agents for each value of k. The dashed red and green envelope curves show the maximum ($R_{\max}[k]$) and minimum ($R_{\min}[k]$), respectively, while the dashed blue curve shows the average ($R_{\text{mean}}[k]$). The accumulated envelopes for the maximum ($R_{\max}[\leq k]$) and minimum ($R_{\min}[\leq k]$) are shown in solid dark red and dark green curves respectively, and the solid dark blue curve shows the accumulated average ($R_{\text{mean}}[\leq k]$).

simpler) policy can be found. Intuitively, the environment would look even easier. In general, focussing on just one value is too risky if we plan to do a robust approximation of the value of difficulty, especially taking into account that in practice we will be forced to analyse samples of the agent population Ω, and not the whole population itself.

Hence, we see next how to analyse $R(\pi)$ against the complexity of π.

3.1 Graphical analysis and indicators

In order to study $R(\pi)$ in terms of the complexity of π, we define *slices* of the whole distribution (conditional distributions actually) according to its complexity. Given a class of agents Ω, we define:

$$\Omega[k] \triangleq \{\pi \in \Omega : K(\pi) = k\}$$

From here, we can parametrise R as follows:

Definition 2 The aggregated reward multi-set per difficulty k is given by:

$$R[k] \triangleq \{R(\pi) : \pi \in \Omega[k]\}$$

By varying k we may construct a series of distributions. We will use the notation $[\leq k]$ to represent all values of complexity from 1 to k. For instance $R[\leq k] = \sum_{j=1..k} R[j]$, known as the 'accumulated'[6] version of $R[k]$.

What does $R[k]$ look like? An example is shown in Figure 2. If we look at the distributions in this figure, we see that the distribution typically widens with increasing values of k. This does not mean that the expected reward increases (it remains constant in this case), but that the higher variability given by more complex programs makes it possible to consider more diverse policies and find some better (and some worse) choices.

We can derive some indicators of difficulty as a summarisation of the distribution. The first idea for difficulty is to consider the maximum. The maximum envelope is just defined as:

$$R_{\max}[k] \triangleq \max_{\pi \in \Omega[k]} R[k] \tag{5}$$

[6] An 'accumulated' distribution is not a *cumulative* distribution.

Clearly, $R_{\max}$, as was already defined (see Eq. 1), can also be calculated as $\max_k R_{\max}[k]$. In the example of Figure 2, this is 10.8. The previous Eq. 5 can be similarly defined for $R_{\min}$ and R_{mean}. By looking at $R_{\max}$, $R_{\min}$ and R_{mean} we can get some information about the environment. If R_{mean} is not centred, it may indicate some kind of bias (i.e., the environment can be benevolent or malevolent, or simply that the rewards are not balanced). This suggests that results may need to be normalised. Normalisation looks important because it makes possible to aggregate the results of one (or more agents) on several environments, by making them commensurable.

If we look for one simple indicator of difficulty, yet again our first approach of a measure of difficulty could be to choose the smallest value of k where $R_{\max}$ is found:

$$\text{argmin}_{k=1..\infty}(R_{\max}[k])$$

which is exactly equivalent to $\hbar$, as seen in Eq. 4. In the environment of Figure 2, we see that the maximum value is reached at complexity 30, so we have that $\hbar = 30$. However, there are other policies with almost-as-good results at complexity 14. This shows that this indicator is not robust. As an alternative, we could also calculate the smallest value of k for a given value of R or figure out other indicators. Nonetheless, any indicator based on a single number is prone to problems for real situations, where we can only get a sample of the points, and not all of them. Yet another option is to calculate a quantile of the data, e.g., 99%, and get the lowest value of k such that we get 1% of the data above a certain value (such as a percentage of the maximum, e.g., 95% of $R_{\max}$). Many other statistical indicators can be used, including a comparison of the distributions. For instance, we could determine for which value of k the sliced distributions stabilise in order to determine when a more robust maximum has been reached.

Another interesting indicator is to calculate the correlation between reward and complexity. If we consider the correlation of all the values, i.e., $Cor_{\pi\in\Omega}(K(\pi), R(\pi))$, we should expect to have no correlation (as in Figure 2). Quite differently, we can calculate the correlation by slice using the maximum value, i.e., $Cor_{i=1..k_{\max}}(i, R_{\max}[i])$ (with $k_{\max}$ being the maximum complexity which is considered). Here, a high positive correlation is expected. In Figure 2, this is 0.73 (Spearman correlation). We will see the use of some of these indicators in section 4.

Nonetheless, the use of the envelope (either as given by the maximum points or by a quantile of the data) may be insufficient to capture the difficulty that an agent finds to get good rewards in general. This issue is related to how the points are distributed for each slice, which is, in turn, related to the notion of discriminating power. Discriminating power was initially stated, in an informal way, as a property of problems or tasks that can be solved by some agents but not by others, i.e., showing high variability in results. An interesting plot for understanding this would be to show the distribution for each slice, or the distribution of R for the whole data. The degree and type of dispersion (measured by kurtosis or other indicators) could be useful here. For instance, if the values between $R_{\max}$ and $R_{\min}$ are distributed almost uniformly we have that there are many policies closer to the optimal result. If it has a more peaked shape (for instance like a beta distribution with $\alpha = \beta = 3$), then the extreme values may be more difficult to reach. Summing up, while this traditional analysis of a distribution may give clear findings occasionally (we will look at several cases in sections 4 and 5), the approach may fail, in general, to disentangle difficulty and discriminating power. Let us look for a different approach next.

3.2 Policy-searching strategies

The picture that is portrayed in Figure 2 gives us information about the location of each policy, but this policy landscape may be still explored in different ways. For instance, we expect that any method trying to solve a problem (e.g., achieving high rewards in an environment) will use a particular traversal strategy in this landscape: some policies will be explored first, because the method has some a priori expectation or knowledge of them being good solutions. In other words, the picture of how policies distribute in terms of their complexity and their expected rewards is very informative, but this needs to be complemented with the the way these policies are going to be explored or sampled from these infinitely large pool of policies. While setting an exploration order is a possible option, we think that the idea of sampling according to an a priori distribution is more general and flexible.

Definition 3 Given a population of policies, Ω, the *a priori policy probability*, denoted by $w(\pi)$, is defined as a distribution over all $\pi \in \Omega$.

Sampling	Distribution	Memory	Related search methods
With replacement	Unaltered	Not Required	Monte Carlo
Without replacement	Updated	Required	Levin's optimal search

Table 1 Prototypical policy-searching strategies according to solver's memory. The sampling with replacement assumes no memory between attempts and it boils down to a Monte Carlo approach using the a priori distribution. The sampling without replacement represents a (more realistic) method that takes into account previous results, and updates the distribution accordingly. If the original distribution is inversely related to the complexity of the policy, then this strategy resembles Levin's optimal search.

Strategy	Environment Reset
Fresh	Yes
Chained	No

Table 2 Prototypical policy-searching strategies according to the problem's memory. The *fresh* strategy assumes the environment is reset between attempts (environment has no memory between attempts). The *chained* strategy represents cases where there is only one attempt and not several episodes that can be reset, so several policies must be tried on a single, long episode.

This gives more or less weight to policies and can represent the probability of finding, using or exploring a policy. We want this w to be as objective as possible. When Ω is finite, a uniform distribution might be a choice. However, when Ω is infinite, this uniform distribution is not possible. In either case, when policies are expressed using a language, the size of the description of each policy (in terms of the constructors involved or the size of the algorithm) is more relevant, as policies that require the use of many constructs or the execution of a long algorithm will be more difficult to find. This is in fact what we have argued for the way that results have been sliced by k in the previous section. This suggests that a good choice for $w(\pi)$ in general may be $w(\pi) = 2^{-K(\pi)}$, with K being a measure of complexity (e.g., a variant of Kolmogorov complexity), and properly normalising to get a probability distribution. More formally, if we have a set of policies Ω, the universal distribution would be defined as follows:

$$w(\pi) \triangleq \frac{2^{-K(\pi)}}{\sum_{\pi' \in \Omega} 2^{-K(\pi')}} \tag{6}$$

This probability represents that simple policies will be explored with much higher probability. This is a reasonable choice as an a priori probability. If we are concerned about the computational complexity of executing a policy (as this may be relevant for many searching strategies), then a better option would be to use Levin's Kt complexity [48] or any other variant that combines program size and execution time. This choice would make the distribution closely related to Levin's optimal search [48], which is known to asymptotically dominate any other search.

While this gives us a reasonable choice for $w(\pi)$, this just defines the a priori distribution, but this probability may (or may not) be updated as the policy space is explored. For instance, if we have explored a policy and it does not give good results, it seems reasonable to assign very low (or zero) probability to that policy, so that the policy is not attempted again (like a sampling without replacement). This is only possible if the strategy has *memory*, and we can recall all the policies that have been tried so far (updating the probability is one way of having memory). Other methods may have no memory (or a limited one) and sampling with replacement would be a better model of the situation. Also, some methods may occasionally ignore the probability distribution and act (almost) randomly for a while, e.g., to get more information from the environment, depending on their exploration-exploitation philosophy. This leads to many possibilities, as many as possible methods can be conceived. However, we can identify some prototypical combinations of the above factors as shown in Table 1, according to the solver's memory.

Finally, another question when exploring several policies is whether the method is allowed to try each policy from scratch ('fresh') or must continue from the state the previous policy left the environment ('chained'). In the case of environments, the exploration of a policy may lead the agent to bad or unresponsive states. The evaluation of further policies would be altered (or even impeded) if this is the case. For instance, a real prey-predator situation may not give the agent a second chance. Table 1 summarises the two prototypical cases, according to the problem's memory.

In many single-agent systems, sampling with replacement is usually associated to a *fresh* strategy (like assuming that both agent and environment have no memory between attempts). Nonetheless, in other cases, especially in multi-agent systems, this association does not always hold, and we can have a *fresh* strategy with or without replacement. We will see some of these options in sections 4 and 5.

We will use the general notation $\Omega||_{w,N}$ for a sample S of size N from the set Ω by using the probability $w(\pi)$, and we will specify whether it uses replacement or not in each case.

3.3 Environment response functions

As we mentioned in section 2, item response theory (IRT) [19] is a paradigm for the study of items (tasks) and a more principled way of designing tests and other instruments that measure abilities, especially in the area of (computerised) adaptive testing. The most distinctive feature about IRT is that each item (task) is categorised and analysed according to its difficulty in terms of the response that subjects of different ability degrees may show at the task.

We saw an example of the three-parameter logistic model and a linear model in Figure 1. Typically, for tasks which are scored by continuous but bounded rewards, the curves should look something in between of the two models. Nonetheless, we do not even need to define a parametric model: what we really need is to determine a function that returns the expected reward given an ability level. In order to do this, we can look at our policy (agent) distribution again and sample from it.

We want to explore the probability that a random sample according to the distribution w achieves a good solution. In order to admit good, but suboptimal solutions, we will consider a tolerance value γ, with $0 \leq \gamma \leq 1$. This gives a positive and negative target: $r_{pos}(\gamma) = (1-\gamma)(R_{\max} - R_{\text{mean}}) + R_{\text{mean}}$ and $r_{neg}(\gamma) = \gamma(R_{\text{mean}} - R_{\min}) + R_{\min}$. This leads to the following definition:

Definition 4 Given a random sample according to w and a tolerance value γ, the probability that the sample of size N contains a value sufficiently close (γ) to the maximum aggregated reward of the population Ω is given by:

$$q_{\text{pos}}(\gamma, \Omega, w, N) \triangleq \Pr\left[\max_{\pi \in \Omega||_{w,N}} \{R(\pi)\} \geq r_{pos}(\gamma)\right] \tag{7}$$

We can define q_{neg} in a very similar way to q_{pos}:

$$q_{\text{neg}}(\gamma, \Omega, w, N) \triangleq \Pr\left[\min_{\pi \in \Omega||_{w,N}} \{R(\pi)\} \leq r_{neg}(\gamma)\right] \tag{8}$$

Note that the maximum (or minimum) is unique but there might be more than one policy reaching that maximum (minimum). For instance, if we make a sample of size 1, we have that:

$$q_{\text{pos}}(\gamma, \Omega, w, 1) = \sum_{\pi \in \Omega} w(\pi) : R(\pi) \geq r_{pos}(\gamma) \tag{9}$$

From this definition, we can show that there is a baseline probability, even for the extreme tolerance case $\gamma = 1$:

Proposition 1 *If R_{mean} is equal to the median of the aggregated reward in Ω and $w(\pi)$ is independent of $R(\pi)$ then $q_{pos}(1, \Omega, w, 1) = 1/2$.*

Similar results as proposition 1 can be obtained for q_{neg}. Proofs can be found in the appendix.

For the case with replacement (e.g., a *fresh* strategy)), equations 7 and 8 can be obtained directly. For instance, the positive case leads to $q_{\text{pos}}(\gamma, \Omega, w, N) = 1 - (1 - q_{\text{pos}}(\gamma, \Omega, w, 1))^N$. However, this does not simplify to a short closed-form expression in the case without replacement. Nevertheless, it is obvious that q_{pos} without replacement is higher or equal than q_{pos} with replacement, so this expression can be used as a lower bound for both cases.

We now look for the minimum size of the sample such that the above probability (either q_{pos} or q_{neg}) is at least 1/2. In other words, we want to know how large the sample has to be to have probability 1/2 (or larger) of getting the maximum (up to a tolerance level):

$$N_{\text{pos}}(\gamma, \Omega, w) \triangleq \min\{N : q_{\text{pos}}(\gamma, \Omega, w, N) \geq 1/2\} \tag{10}$$

$$N_{\text{neg}}(\gamma, \Omega, w) \triangleq \min\{N : q_{\text{neg}}(\gamma, \Omega, w, N) \geq 1/2\} \tag{11}$$

We can see that with replacement, $N_{\text{pos}}(\gamma, \Omega, w) = \frac{1}{-log_2(1-q_{\text{pos}}(\gamma,\Omega,w,1))} + 1$, by just setting the above expression for q_{pos} with replacement to be less than or equal to 1/2.

From here, we have:

Proposition 2 *The minimum size of the sample to have 0.5 probability of obtaining the maximum reward with tolerance $\gamma = 1$ and $\gamma = 0$ is given by:*

$$\begin{aligned} &N_{pos}(1, \Omega, w) = 1 \\ &N_{pos}(0, \Omega, w) = \begin{cases} |\Omega|/2 & \textit{(without replacement)} \\ \frac{1}{-log_2(1-q_{pos}(\gamma,\Omega,w,1))} + 1 & \textit{(with replacement)} \end{cases} \end{aligned} \tag{12}$$

For the second equality without replacement we also assume the same conditions of proposition 1 and that the policy that produces R_{max} is unique. Otherwise, the upper bound would be smaller. We also assume that Ω is finite. For the third equality we assume that sampling is with replacement, but this is also useful as an upper bound for the case without replacement, as the latter require fewer draws.

This can be done similarly for N_{neg}. The cases of γ between 0 and 1 are more interesting. This means that we need to estimate N_{pos} (and N_{neg}) in order to have a function of the expected value of the number of policies that we need to explore to approach a given reward value.

From proposition 2, we see an important difference between the cases without and with replacement, as the use of memory is useful to exhaust the options when Ω is finite. Even when Ω is infinite, we can have very different results. For instance, if there is a bad policy π_{bad} with $w(\pi_{bad})$ close to 1, then sampling with replacement may try this policy again and again, which is clearly unrealistic. However, if the second highest policy is good and much higher than the other policies, this would be likely obtained on the second draw, making a huge difference in this particular case.

Depending on the choice of w, the functions N_{pos} and N_{neg} will have different shapes. Since Ω, the set of agent policies, can be very large (or even infinite), and this set is discrete, we need to consider a distribution that is appropriate for infinitely many discrete objects. We argued that the universal distribution, as defined in Eq. 6 is a good choice for w. Note that with this distribution (and many other distributions for infinite sets), most of the distribution mass may fall on a few examples. Clearly, sampling with replacement will draw these elements again and again.

Nonetheless, let us assume a universal distribution and get (the previous w in Eq. 10), so we obtain the following property:

Proposition 3 *Consider π^*_{pos} as any of the shortest policies with maximum reward, i.e., any element in the set $argmin_\pi\{K(\pi) : R(\pi) = R_{max}\}$. We have that:*

$$N_{pos}(0, \Omega, w) \leq 2^{K(\pi^*_{pos})} \sum_{\pi' \in \Omega} 2^{-K(\pi')} = 2^{k^*} \cdot w_{all}$$

Using the shorthand notation $k^* = K(\pi^*_{pos})$ and $w_{all} = \sum_{\pi' \in \Omega} 2^{-K(\pi')}$. This result is both valid for the cases with and without replacement, as is proved using replacement.

The previous result is also applicable to the shortest policies with minimum reward, π^*_{neg}. In both cases, the expression suggests that in order to have a measure which is commensurable with the value of complexity k derived from a complexity measure[7], we need to apply a logarithm to N (since $\log_2 N \leq k^* + \log_2 w_{\text{all}}$) with w_{all} being a constant which is the same for all agents (actually, it is 1 if K is normalised). This leads us to propose the following metric[8]:

$$D_{\text{pos}}(\gamma, \Omega, w) \triangleq \log_2 N_{\text{pos}}(\gamma, \Omega, w) \tag{13}$$

$$D_{\text{neg}}(\gamma, \Omega, w) \triangleq \log_2 N_{\text{neg}}(\gamma, \Omega, w) \tag{14}$$

[7] Note that this is independent of what kind of measure K we use. This can be an approximation to Kolmogorov complexity or any of its variants, or simply the length of the policy in a given language.

[8] Both expressions can be simplified in the case with replacement. For instance, $D_{\text{pos}}(\gamma, \Omega, w) = \log_2 \log_2(1 - q_{\text{pos}}(\gamma, \Omega, w, 1))$.

Assuming we fix some given w and Ω, the functions of difficulty can be expressed solely in terms of γ, i.e., $D_{\text{pos}}(\gamma)$ and $D_{\text{neg}}(\gamma)$. Note that, from propositions 1 and 2, we have that $D_{\text{pos}}(1) = \log_2 1 = 0$, i.e., the difficulty with tolerance $\gamma = 1$ is just 0. Similarly for D_{neg}. The maximum value of difficulty under independence of $w(\pi)$ and $R(\pi)$ and tolerance $\gamma = 0$ would be $\log_2 |\Omega|/2 = (\log_2 |\Omega|) - 1$, when sampling is without replacement. This sets a range of difficulty between 0 and ∞ for infinite sets. For finite sets, the upper limit may be lower than $\log_2 |\Omega|$, if there is positive dependence between $w(\pi)$ and $R(\pi)$. In fact, this value could even be finite for infinite sets Ω if a strong correlation exists. Of course, with different values of γ, we would get difficulty values between the minimum and the maximum.

Now, if we are able to estimate D_{pos}, mapping tolerance γ onto a value of complexity k, we can derive $(D_{\text{pos}})^{-1}$, which returns a value of γ that ensures 1/2 probability of finding that level of aggregated reward using a sample size related to 2^k. Instead of interpreting the input k as a difficulty, we can see this as an ability θ of the agent, leading to a curve very much like an item response curve. Also, we can just translate the output of $(D_{\text{pos}})^{-1}$ to an aggregated reward by applying the resulting tolerance expression, and defining:

Definition 5 The *positive environment response function* (or curve) is defined as:

$$\Re_{\text{pos}}(\theta) \triangleq (1 - (D_{\text{pos}})^{-1}(\theta))(R_{\text{max}} - R_{\text{mean}}) + R_{\text{mean}} \text{ for } \theta \geq 0.$$

with output between R_{mean} and R_{max}. This non-decreasing function can be plotted in the form of *environment response curves*. We can do similarly for D_{neg} leading to

Definition 6 The *negative environment response function* (or curve) is defined as:

$$\Re_{\text{neg}}(\theta) \triangleq (D_{\text{neg}})^{-1}(-\theta)(R_{\text{mean}} - R_{\text{min}}) + R_{\text{min}} \text{ for } \theta \leq 0.$$

with output between R_{mean} and R_{min}. Putting both things together gives the following function:

Definition 7 The *environment response function* (or curve) is defined as:

$$\begin{aligned}\Re(\theta) \triangleq\ & \Re_{\text{neg}}(\theta) \text{ if } \theta < 0 \\ & \Re_{\text{pos}}(\theta) \text{ otherwise}\end{aligned}$$

Note that the above curves are normalised on the x-axis (several agents can be compared by their ability values on different problems) but are not normalised on the y-axis, because we recover the original scale given by R_{max}, R_{min} and R_{mean}. By just setting these terms to 1, -1 and 0 respectively, we can attain a normalised environment response curve. An illustrative example is shown in Figure 3.

We have used θ for the ability, as in Item Response Theory. The use of a negative ability may look strange initially, but it has a good interpretation. We can see the value of $\theta = 0$ as the expected ability for all policies. So, performing worse than this can only happen by chance or because the agent is making an effort to get bad rewards. This leads to a duality in these plots.

4 Case Study: An agent-populated elementary cellular automaton

In theory, we can analyse difficulty and discriminating power for any possible kind of environment which is compatible with definition 1, i.e., any interactive environment or task. In this section, we are going to choose a simplistic setting for practical reasons. First, it is more illustrative to use minimalistic environments where the number of observations and actions are extremely reduced, while still having some relatively rich phenomena with very simple transition functions. Second, we are interested in simplistic policy languages in order to be able to evaluate a large amount of agents quickly. Third, we want to elaborate on settings that are well known and previously studied in terms of emergence and complexity. The configuration we present next (based on a hybridisation of cellular automata and multi-agent systems, which has been proposed as a way of uniting these two areas [27]) follows these three conditions.

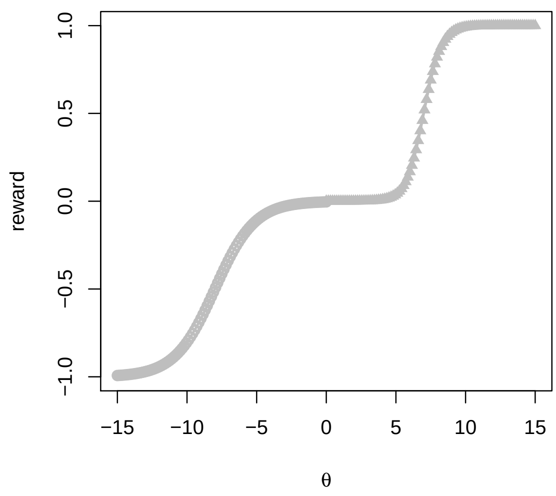


Fig. 3 Example of a normalised *environment response curve*. Given an expected ability θ on the x-axis we get an expected aggregated reward on the y-axis. If we only focus on the positive (top right) quadrant of the plot, we can see that the plot saturates at about $\theta = 8$. For different values of tolerance γ (that could be seen as percentages between 0 and 1, and -1 and 0, on the y-axis), we can also determine how much ability is needed. The bottom left quadrant of the plot shows that any environment is dual, and by negating its rewards we could have an environment behaving as it is shown in the bottom left part of the plot.

4.1 Agent-populated elementary cellular automata: definition and examples

The environments we will work with are composed of an elementary cellular automaton (ECA) [61] for the space $\mathcal{S}$ and the transition function τ, but we will let the agent see and modify part of the usual behaviour of the automaton. The following definition specifies the complete behaviour of this kind of environment:

Definition 8 A single-agent elementary cellular automaton (SAECA) is a special kind of environment $\langle S, \tau, \omega, \rho, \Pi \rangle$, as seen in definition 1, with the following extra parameters $\langle \boldsymbol{\sigma}^0, \nu, p^0 \rangle$. The state space $\mathcal{S}$ is represented by a one-dimensional array of bits or cells $\boldsymbol{\sigma} = \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle$, also known as *configuration.* We consider the array to be finite ($|\boldsymbol{\sigma}| = m$) but circular in terms of neighbourhood ($\sigma_0 = \sigma_m$ and $\sigma_{m+1} = \sigma_1$). There is an initial array $\boldsymbol{\sigma}^0$, also known as *seed.* The transition function τ is given by ν, as any of the $2^{2^3} = 256$ rules that can be defined looking at each cell and its two neighbours according to the numbering scheme convention introduced in [61]. Given the behaviour of the space, we consider just one agent in Π. The agent is located at one cell (its position p) with $1 \leq p \leq m$, which is initially p^0. The set of observations $\mathcal{O}$ is given by two bits $\langle \sigma_{p-1}, \sigma_{p+1} \rangle$ representing the contents of the left and right neighbouring cells respectively, i.e., σ_{p-1} and σ_{p+1}. The actions $\mathcal{A}$ are given by a 'move' and an 'upshot', denoted by the pair $\langle V, U \rangle$. The ordered set of moves is given by { `left`=0, `stay`=1, `right`=2 }, and the ordered set of upshots is { `keep`=0, `swap`=1, `set0`=2, `set1`=3 }, which respectively mean that the content of the cell where the agent is does not change, the content of the cell is swapped ($0 \to 1$, $1 \to 0$), the content is set to 0 and the content is set to 1. The rewards are calculated in the following way. If the agent is at position p at time t, then we use this formula:

$$r^t \leftarrow \sum_{j=1..\lfloor m/2 \rfloor} \frac{\sigma^t_{p+j} + \sigma^t_{p-j}}{2^{j+1}}$$

which counts the number of 1s which are in the neighbourhood of the agent, weighted by their proximity. It is easy to see that $0 \leq r^t \leq 1$.

The order of events for each step in the system is: observations are produced, actions are performed, the automaton is updated and finally, rewards are produced.

Note that the environment is parametrised by the original contents of the array σ^0 (including its size), the ECA rule number ν, and the original position of the agent p^0. Given an environment and a computable agent, the evolution of the system is computable and deterministic.

In order to explore the results for different policies, we need a language for expressing them, APL:

Definition 9 The agent policy language APL is given by a memory (or history) binary array *mem*, initially empty (and not circular), and an ordered set of instructions $\mathcal{I} = \{$ `back`=0, `fwd`=1, `Vaddm`=2, `Vadd1`=3, `Uaddm`=4, `Uadd1`=5 $\}$. The numbers on the right will be used as shorthand for the instruction. For instance, the string 22142335 represents a program in APL. A program or policy π is a sequence of instructions $\iota_1, \iota_2, ..., \iota_{|mem|}$ in $\mathcal{I}$. The interpreter works on its memory by using two accumulators V and U, and the action is given by the result of the accumulators at the end of the process. Namely:

```
 1. Read the observation $\langle \sigma_{p-1}, \sigma_{p+1} \rangle$ and append it to the history array $mem$.
 2. Place the memory pointer $b$ at the end of $mem$.
 3. $V \leftarrow$ stay
 4. $U \leftarrow$ keep
 5. forall $\iota \in \pi$
 6.   case $\iota$:
 7.     back  : $b \leftarrow \max(b-1, 1)$
 8.     fwd   : $b \leftarrow \min(b+1, |mem|)$
 9.     Vaddm : $V \leftarrow (V + mem_b) \bmod 3$
10.     Vadd1 : $V \leftarrow (V + 1) \bmod 3$
11.     Uaddm : $U \leftarrow (U + mem_b) \bmod 4$
12.     Uadd1 : $U \leftarrow (U + 1) \bmod 4$
13.   end case
14. endfor
15. return $\langle V, U \rangle$
```

Let us see an example. If an agent is located at the fifth position of the configuration 000110111 and has a current history $mem = 111010$ then the observations 1 and 0 will be appended to *mem*, leading to $mem = 11101010$. If the policy 20242335 is applied, we start with $b = 8$, $V = 0 =$ `stay` and $U = 0 =$ `keep`, and we have the following execution:

1. $\iota_1 = 2 =$ `Vaddm`, $V \leftarrow (V + mem_8) \bmod 3 = 1 =$ `stay`.
2. $\iota_2 = 0 =$ `back`, $b \leftarrow max(8-1, 1) = 7$.
3. $\iota_3 = 2 =$ `Vaddm`, $V \leftarrow (V + mem_7) \bmod 3 = 2 =$ `right`.
4. $\iota_4 = 4 =$ `Uaddm`, $U \leftarrow (U + mem_7) \bmod 4 = 1 =$ `swap`.
5. $\iota_5 = 2 =$ `Vaddm`, $V \leftarrow (V + mem_7) \bmod 3 = 0 =$ `left`.
6. $\iota_6 = 3 =$ `Vadd1`, $V \leftarrow (V + 1) \bmod 3 = 1 =$ `stay`.
7. $\iota_7 = 3 =$ `Vadd1`, $V \leftarrow (V + 1) \bmod 3 = 2 =$ `right`.
8. $\iota_8 = 5 =$ `Uadd1`, $U \leftarrow (U + 1) \bmod 4 = 2 =$ `set0`.

After this program, which is run internally, we obtain the action that the agent will perform on the environment, which is given by $\langle V, U \rangle = \langle 2, 2 \rangle = \langle \texttt{right}, \texttt{set0} \rangle$. This means that the agent will move right and set the content of the cell to 0.

While the class of policies generated by this language is infinite, the language is still clearly not universal, and all (finite) programs end. The goal of this language is not to be easily programmable but to be able to express some simple policies that may be useful in the environment.

Let us see a few examples of how these environments and agents work. First, Figure 4 shows the evolution of several environments with seed "010101010101010101010", and several values of ν. We include an *inert* agent, i.e., an agent that does not affect the environment (i.e., an empty policy π) at position p_0= 11. As a result, the matrix after 200 iterations is the same as a classical elementary cellular automaton with each number ν, with patterns that are well-known (see, e.g., [61]).

Things change when we incorporate active agents in the environment. Figure 5 shows how the environment with elementary cellular automaton number 110 varies for several agent policies. The resulting matrix patterns are different. Similar things (where differences are more visible with respect to the corresponding diagram in Figure 4) happen with rule number 164 (Figure 6).

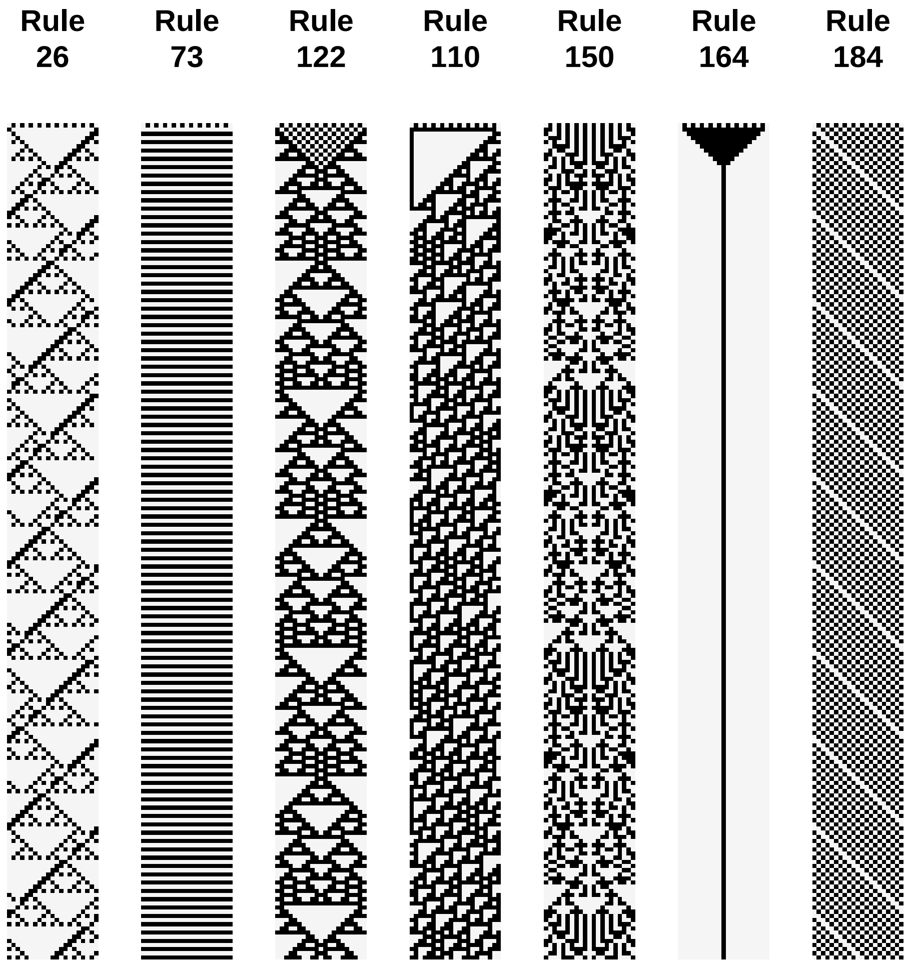


Fig. 4 Space-time diagram (evolution for $t = 200$ steps) of several elementary cellular automata without agent (or with an agent with a policy that always does actions of the form $\langle \texttt{stay}, \texttt{keep} \rangle$, as the empty policy). The initial array (seed) is always 010101010101010101010, whose length is 21 bits.

4.2 Experimental setting

Let us see now how the plots, indicators and curves introduced in section 3 can be applied to this particular environment class. To this end, we need to evaluate the population of policies Ω which originate from APL against several environments. Of course, in order to make this evaluation finite, we set a fixed number of steps t and we work with a finite sample of policies $\hat{\Omega}$.

As we mentioned in section 3.1, we consider that systems and approaches to solve a problem usually explore many possible solutions or policies through policy-searching methods. Although there are infinitely many policy-searching methods (and each of them actually constitutes a policy as well), in Table 2 we chose two representative options. In the *fresh* case we start as if the environment were brand new, so the seed configuration is reinitialised for each policy. In the *chained* case we evaluate them sequentially, without resetting the environment to its initial state.

The *chained* strategy is sensitive to cases where the environment is ‘spoiled’ by a bad policy and set into an unresponsive state such that rewards cannot be modified any more. If this is the case, this will be shown in the distribution plots. For instance, in some ECAs, a previous policy may leave the configuration without 1s and the environment will not be able to easily recover from there.

In what follows, we show results with the *fresh* strategy and the *chained* strategy. In both cases, we interweave a 5% of random-walk policies (note that random-walk policies are non-deterministic and cannot be expressed by the language APL). These random actions will also be useful to get information about any bias of the environment.

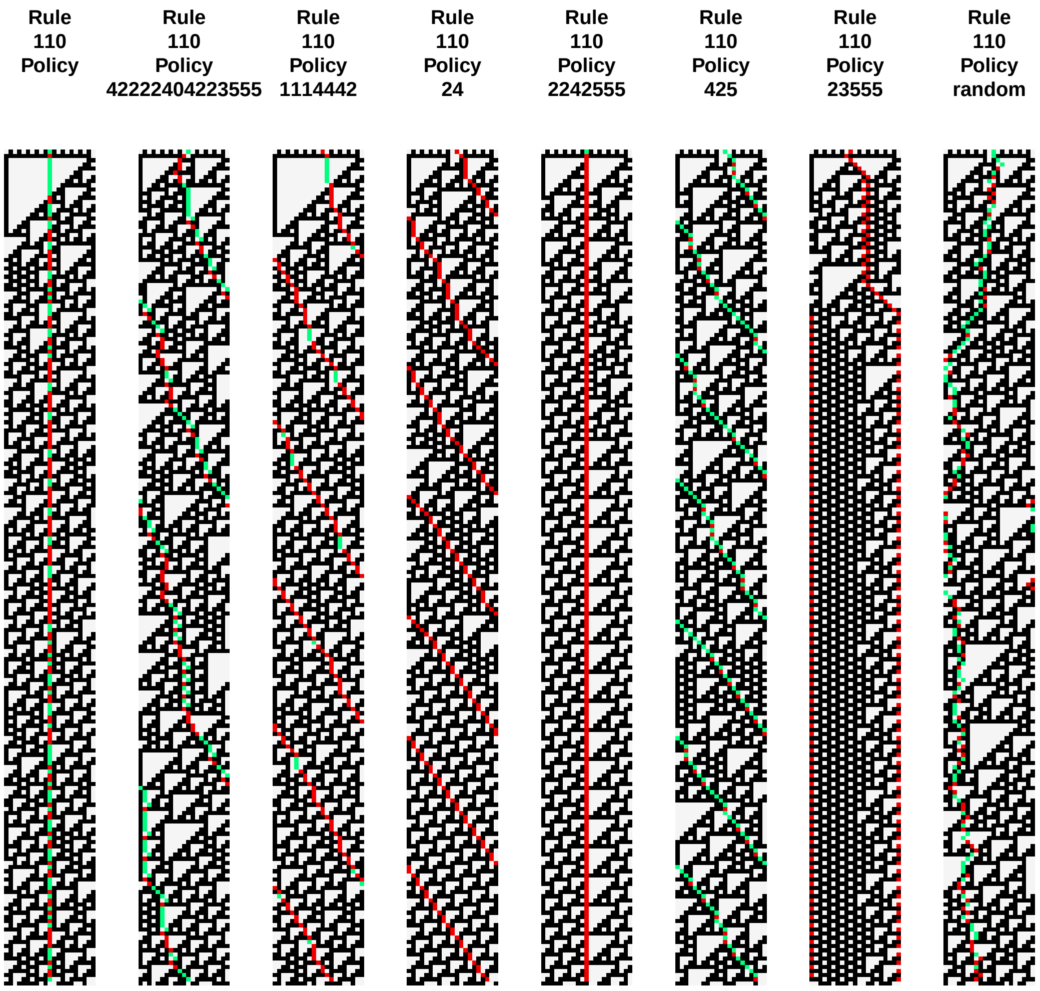


Fig. 5 Space-time diagram (evolution for $t = 200$ steps) with different agent policies for the elementary cellular automaton with rule 110. The initial array (seed) is always 010101010101010101010, whose length is 21 bits. The agent is represented by a red dot when the cell has a 0 (like the white ones) and by a green dot when the cell has a 1 (like the black ones). The leftmost diagram is the empty policy ($\langle$`stay`, `keep`$\rangle$) and the rightmost one is a random-walk policy (it cannot be described with APL).

We now describe more details about the experiments. We set the number of steps t to 300. The seed configuration is always "010101010101010101010" and the start-up agent position is $p_0 = 11$. From the infinite set of policies Ω that can be obtained from the language APL, we generate a working sample $\hat{\Omega} \subset \Omega$ with 2,000 policies. As mentioned above, 100 are random-walk policies (these are interwoven regularly between the others in the *chained* strategy). The remaining 1,900 policies are randomly generated using a uniform distribution for the size of the APL program (between 1 and 20 instructions), so making up about 95 policies each. Given the size of the program, we choose each instruction with a uniform distribution among the six instructions in APL. Except for small lengths, policies are rarely equal by chance, but they can be *equivalent*. For instance, a `back` instruction cancels if followed by a `forward` instruction, many instructions at the end are useless if there is no `Vaddm`, `Vadd1`, `Uaddm`, `Uadd1` after them, as well as some other simplifications. Because of this, we built a simplifier that transforms programs (when possible) into (equivalent) shorter ones. This simplification has no effect at all on how the policies perform. It just affects their length (even leading to policies with no length, performing no action) and, usually, their complexity estimation. The application of the simplifier makes that the frequencies of programs of small size is slightly increased and the number of programs of large size (e.g., > 15) is reduced considerably. This phenomenon has to be taken into account when looking at the plots and results.

Finally, in order to approximate the Kolmogorov complexity for each policy, we applied a compression algorithm to all the programs. In particular, we used a common approach [63, 42]: we coded the program as a character string and compressed it (using the *memCompress* function in R [57], a GNU project implementation of Lempel-Ziv coding). As a normalisation, from the resulting complexity we subtracted

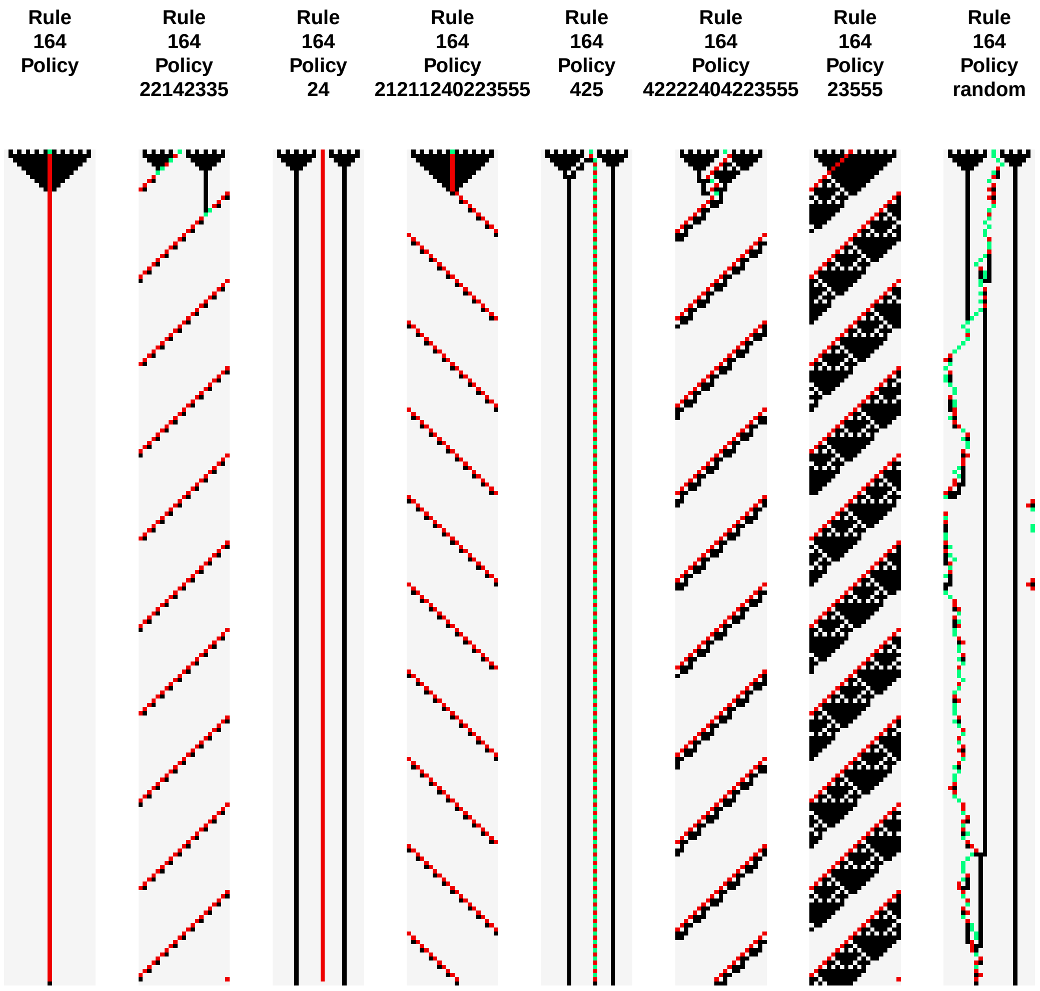


Fig. 6 Same as Figure 5, with rule 164 and other policies.

the length of the compressed size of an empty string. This makes the complexities range between $k_{\min} = 0$ and $k_{\max} = 20$. Given that Lempel-Ziv works with bytes and APL has only 6 instructions, the complexity has to be understood as being multiplied by $\log_2(6) = 2.58$ and not by $\log_2(256) = 8$. In any case, this factor is the same for all policies so it is irrelevant for our study. The important thing is that the joint use of the simplification and the compression processes can give a relatively acceptable (and efficient) approximation of Kolmogorov complexity for these short strings.

4.3 Analysis of the distribution

As we have mentioned above, we performed experiments using a *fresh* strategy and a *chained* strategy, with a 5% of random-walk policies. Figure 7 shows the distributions $R[k]$ for several environments (rules 184, 110, 122, 164) using the *fresh* strategy. We show complexity (k) on the x-axis and aggregated reward (R) on the y-axis. In the figure, each small circle (in grey) shows a policy. The box plots and the envelopes are shown for the accumulated values ($[\leq k]$). As mentioned above, there is a higher concentration of values of k between 5 and 10, because the uniform distribution between 0 and 20 for program length becomes compacted by the simplifier and the calculation of (an approximation of) Kolmogorov complexity.

We see that the plots are very different in terms of average, range and evolution. The environment with rule 110 evolves very quickly towards the maximum envelope, while rule 122 has a slower trend. The environment with rule 184 has very a wide range and a high dispersion. On the contrary, rule 164 has a small range (especially for rewards below the average). There are also very important differences

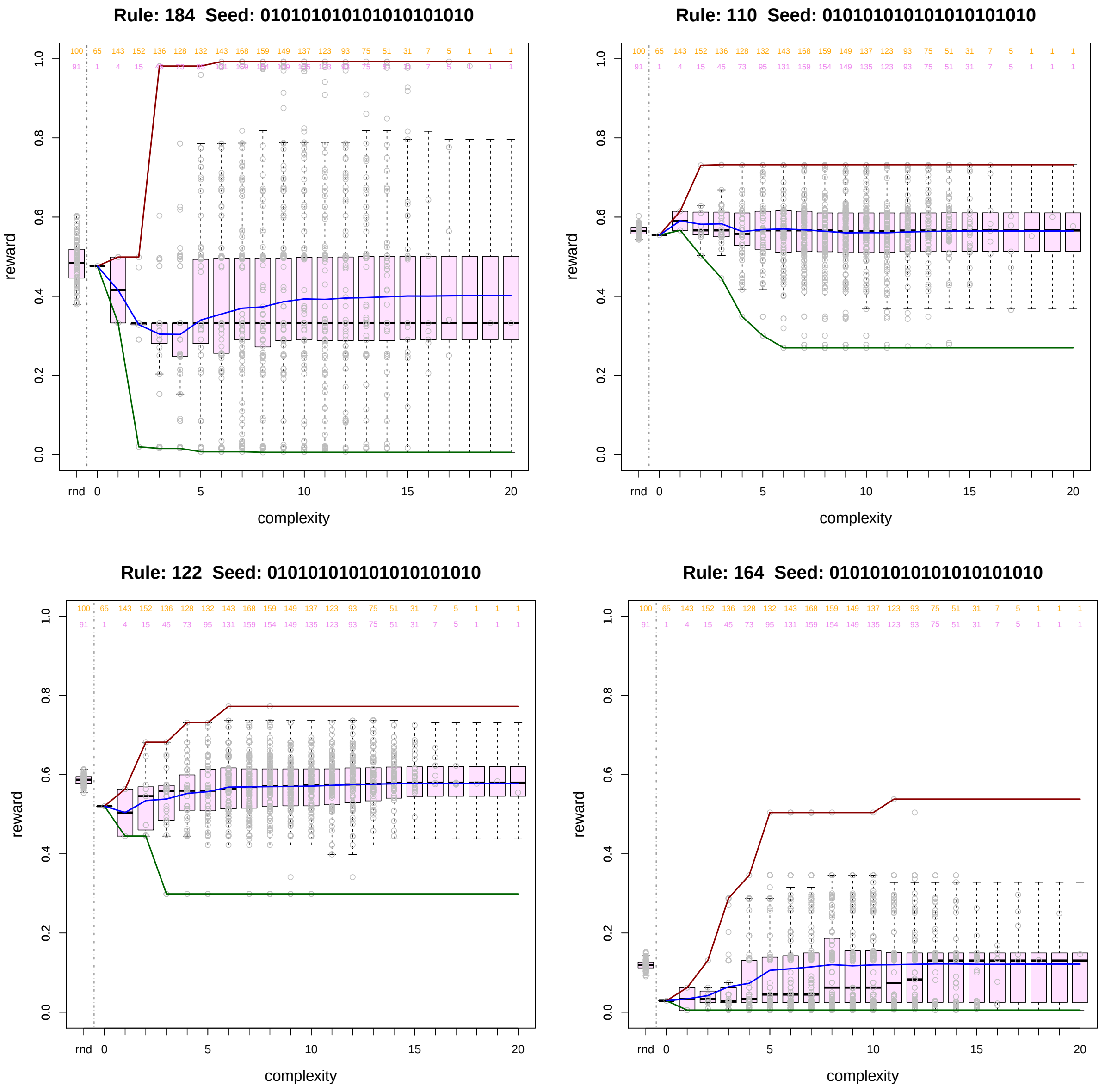


Fig. 7 We show the distributions $R[k]$ for several environments (rules 184, 110, 122, 164) according to the complexity of the policy. We generate 2,000 agents (100 of them are random) and for each of them the environment runs during 300 iterations ($t = 300$), starting from scratch for each agent (*fresh* strategy), using the same seed configuration all the time: '01010101010101010101010'. Some generated agents are equivalent. This is shown in the top part of the plots: in orange (first row) we have the number of agents of that complexity that we generated, in pink (second row) just counting the ones that are different. Note that this is just for information, the policies are not actually removed, but considered as many times as they appear (to preserve the original distribution). On the left of the plots (indicated on the x-axis with "rnd") we also show the result for 100 random-walk policies. For each value of k (x-axis) we show the aggregated reward results (y-axis) for each policy (in grey, as small circles) and a summarised representation of the distribution using the accumulated values ($[\leq k]$) for the box (and-whiskers) plots. Each box shows the lower and upper quartiles, the band showing the median and the ends of the whiskers showing 1.5 of the interquartile range (if this does not exceed the maximum and minimum). The dark red and dark green lines show the maximum and minimum envelopes and the blue line shows the average for $[\leq k]$ (hence their monotonic modified[behaviour]). Each environment shows a different figure, which provides insight about the difficulty and discriminating power of each environment. In all cases, note the low dispersion of the random-walk policy.

in terms of average aggregated reward (R_{mean}) as well. Some environments show some clusters. For instance, the environment with rule 184 shows a cluster of values around $R = 0.95$, as well as at 0.7, at 0.5, 0.3, and 0.05. There are some areas that are almost empty, even for high values of k, such as the range between 0.8 and 0.9. Finally, the results for the random-walk policies (shown on the left of the plots) are usually very compact, showing that an analysis using random actions (instead of policies) is not able to show the structure of the problem.

Rule	$\lvert\hat{\Omega}\rvert$	$R_{\max}$	R_{mean}	$R_{\min}$	$\hbar$	$Cor_{\pi\in\hat{\Omega}}(K(\pi),R(\pi))$	$Cor_{i=1..k_{\max}}(i,R_{\max}[\leq i])$
184	1900 (1349)	0.97	0.46	0	3	0.08	0.69
184	9500 (5922)	0.99	0.46	0	5	0.06	0.80
110	1900 (1349)	0.74	0.57	0.28	7	0.01	0.88
110	9500 (5922)	0.81	0.58	0.27	15	-0.02	0.98
122	1900 (1349)	0.80	0.57	0	5	0.16	0.81
122	9500 (5922)	0.82	0.59	0	5	0.11	0.80
164	1900 (1349)	0.50	0.06	0	7	0.11	0.87
164	9500 (5922)	0.50	0.06	0	7	0.07	0.87

Table 3 Comparison of indicators using the *chained* strategy for two different sizes of the working sample $\hat{\Omega}$ (in parentheses the number of policies after removing the repeated policies). Those for $\lvert\hat{\Omega}\rvert = 1900$ correspond to Figure 8. All correlations are rank (Spearman) correlations.

The values for $R_{\max}$ for the four environments (184, 110, 122, 164) are 0.99, 0.73, 0.77 and 0.54, respectively, and the smallest complexities to reach these values ($\hbar$) are 6, 3, 6 and 11, also respectively. The R_{mean} values are 0.40, 0.56, 0.58 and 0.12 respectively, and the $R_{\min}$ values are 0.01, 0.27, 0.30 and 0.01 respectively. The (Spearman) correlations $Cor_{\pi\in\hat{\Omega}}(K(\pi),R(\pi))$ are 0.12, 0.03, 0.17 and 0.09 respectively[9], and the (Spearman) correlations $Cor_{i=1..k_{\max}}(i,R_{\max}[\leq i])$ are 0.84, 0.70, 0.85 and 0.94 respectively. If we take a look at normalisation, we see that the average results for random-walk policies are 0.49, 0.57, 0.59 and 0.12, which are very similar to the R_{mean} values except for environment 184. This suggests that we can devise environments to have 0 expected aggregated reward for a random-walk agent (random *actions*), as with the notion of 'balanced' environments in [34,33, sec. 4.2], or we can normalise the results after execution, in order to get 0 for a random *policy*, as we will do here.

Now we analyse similar experiments with the *chained* strategy in Figure 8. The shapes of the distributions are similar to the previous case, but there are some differences. For instance, the environment with rule 164 is very sensitive to previous policies, suggesting that many policies lead to a low number of 1s from where it is difficult to recover. In fact, the existence of non-recoverable configurations full of 0s cannot be ruled out. Actually, it is symptomatic that the average performance with random-walk agents (action-random) is now higher than with a set of random policies.

We now include some indicators for Figure 8 as well, shown in Table 3. While the values of $R_{\max}$ are similar to the ones for the *fresh* strategy, the $\hbar$ values are very different, confirming that this value is not robust to small changes (in the sample or the strategy). We see some relevant changes with respect to the *fresh* strategy case in $R_{\min}$, which again indicates that some policies may lead to unresponsive environment states from where it is difficult to recover.

We also did some experiments with a larger working sample $\hat{\Omega}$ of 9,500 policies (10,000 minus the random walks). The results are also shown in Table 3. As we can see, while there is a strong similarity in the results for many indicators, for the two different sample sizes, some other indicators vary slightly, and one, $\hbar$, is clearly not robust.

Finally, we show the whole distribution without slicing by complexity in Figure 9. The narrow bars of the histograms (usually higher and placed about in the middle) show the 100 random-walk policies. The wide bars show the remaining 1,900 policies. We see that the histograms for the random-walk policies are peaked in the middle and with a Gaussian shape. Apart from their location, random-walk policies do not provide too much information. However, this is not the case for the other policies. In the case of environments with rules 110 and 122 we see a Gaussian shape, which suggests that, in their ranges, the discriminating power is quite regular, with more policies around the average and very few on the extremes of the distribution. On the contrary, environments with rules 184 and 164 show a very different picture. Rule 184 has a high concentration of policies for aggregated rewards close to 0, 0.5 and 1, and valleys for the rest. This makes it difficult to understand whether the discriminating power is high or low. Also the notion of difficulty is blurred here, because it may seem difficult to reach 0 and 1 because of their location, but then there are many policies achieving these scores. Rule 164 gives a very asymmetric picture, and many policies behave worse than random.

[9] This small bias may be originated because the instruction setting 1s comes before the instruction setting 0s in the APL instruction set.

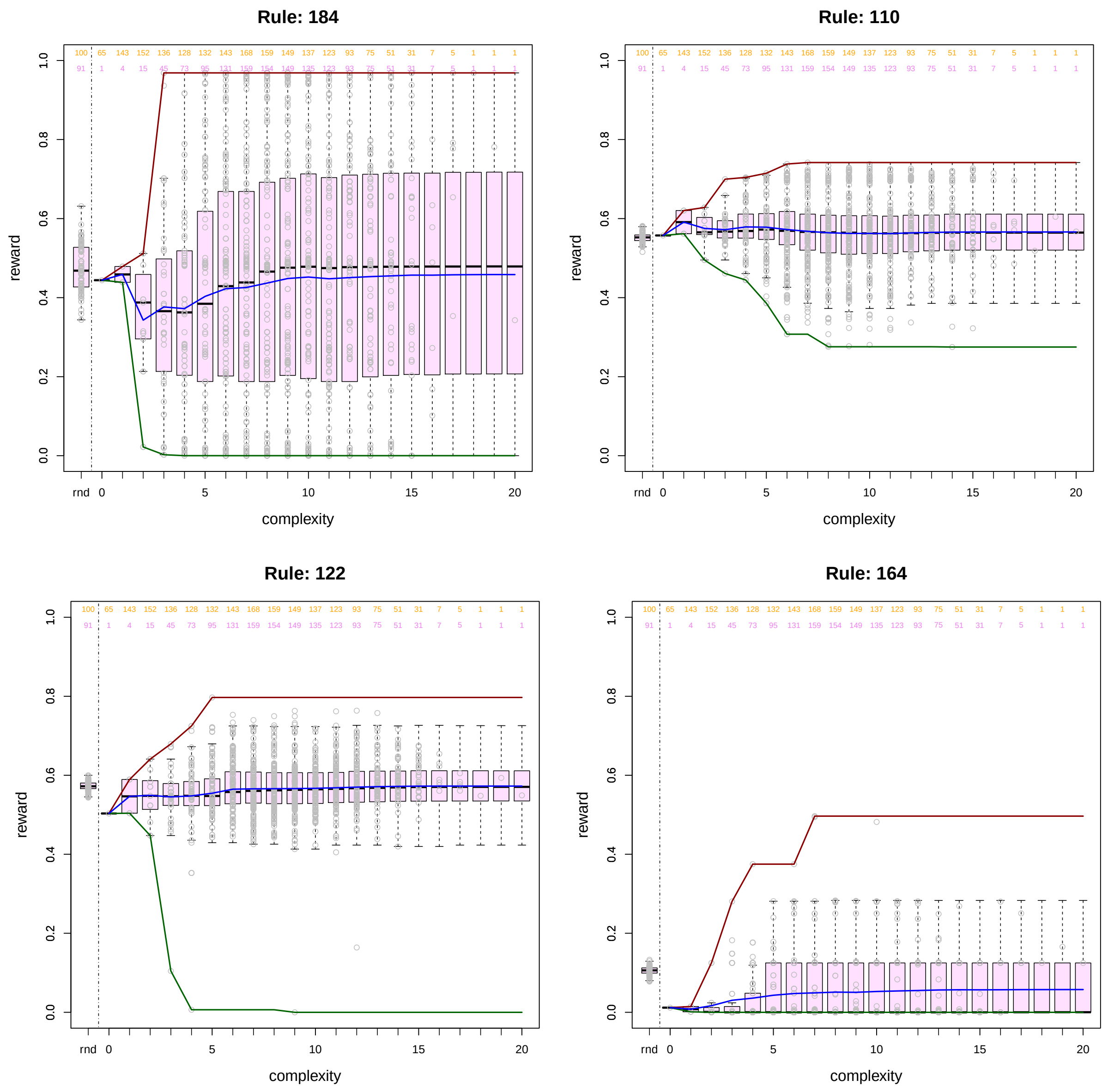


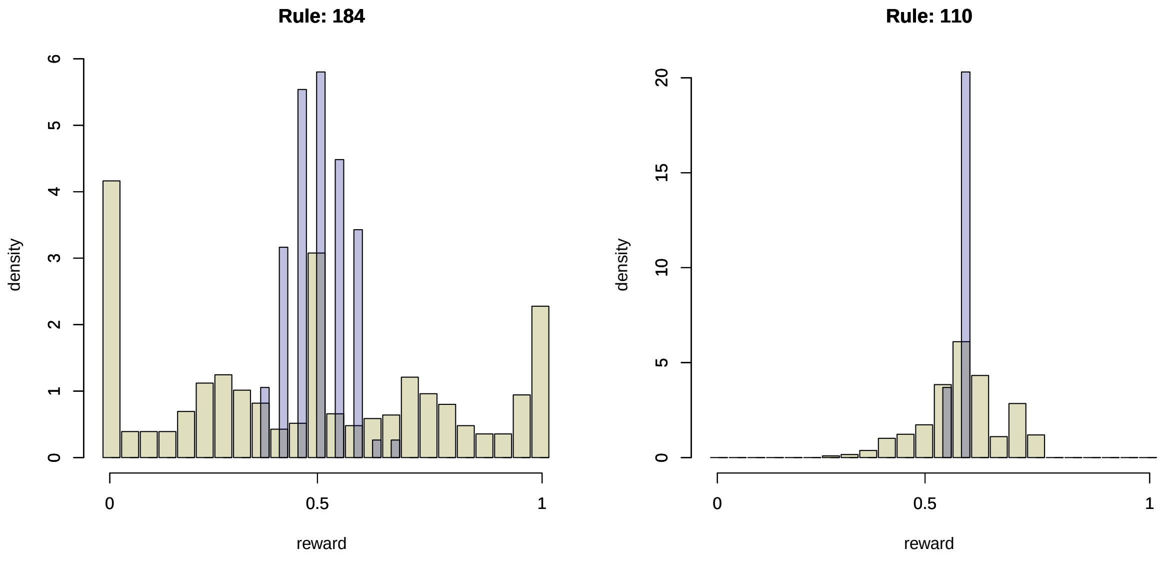
Fig. 8 Same as Figure 7 but using the *chained* strategy for the experiments.

In practice, we need a clearer way of determining the actual difficulty, normalisation and discriminating power of each problem. We introduced the environment response curves for that purpose in section 3.3 and we will see them next for this setting.

4.4 Estimation of the environment response curves for the *chained* strategy

First of all, we give a few indications of how the environment response curves are calculated. We estimate $w(\pi)$ in Eq. 6, by using k as an approximation of $K(\pi)$ and then we sample over the previously built working sample $\hat{\Omega}$, which had 1,900 policies. Since we need to invert the functions D_{pos} and D_{neg}, we calculate 101 different values of γ at equal intervals from 0 to 1.

In section 3.3 we did not find a closed-form expression for the inverted functions when sampling without replacement (see Table 1), which we think is more realistic for this problem than sampling with replacement. So in this case we perform the following procedure. For each value of γ, we construct samples S (without replacement, i.e., $S \leftarrow \hat{\Omega}||_{w,N}$) with values of $N = 1..1900$, and we calculate $\max_{\pi \in S} R(\pi)$ for each sample S. We calculate whether this value is greater than or equal to $r_{pos}(\gamma)$ as in Eq. 7 for



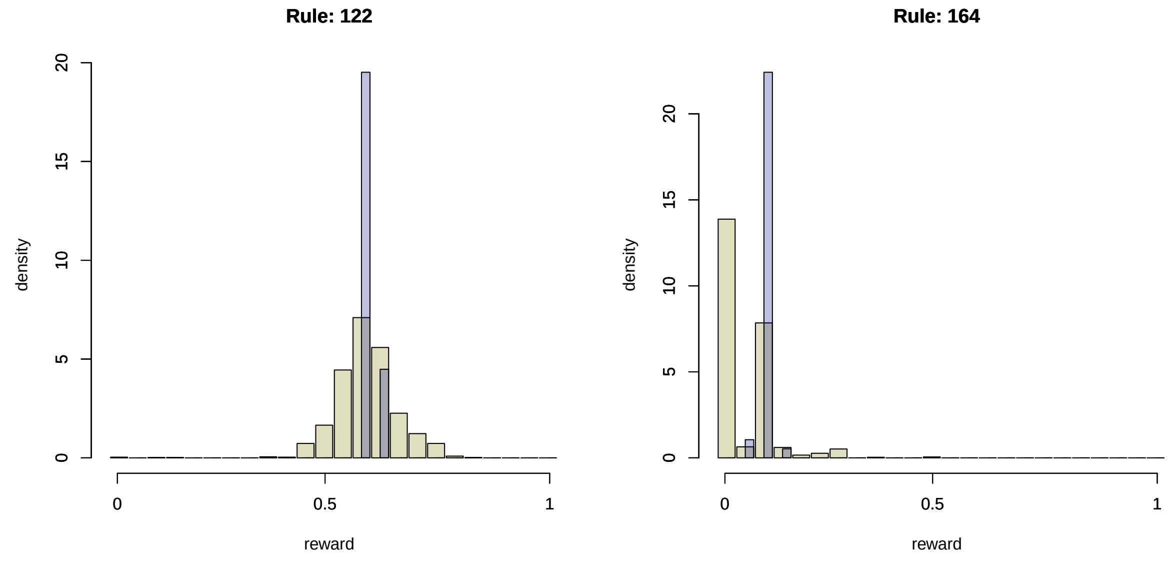


Fig. 9 Histograms of R for the 2,000 policies for the same experiments in Figure 8 (*chained* strategy). Narrow bars (purple) represent all the results for the random-walk agents while wide bars (in beige) represent the rest.

each of these samples, leading to an array of Boolean values denoted by B, whose size[10] is 1,900. Note that this is not an estimation of q_{pos} for each N, but just one case. Since we want to calculate N_{pos} and N_{neg} (see equations 10 and 11), we need to estimate when q_{pos} is greater than or equal to $1/2$. We do this by summing how many *falses* there are in B. This is exactly N_{pos}. Since our value for q_{pos} was not a probability but just one case, we repeat the process 400 times in order to get a good estimate of N_{pos}. Finally, by calculating the binary logarithm we have D_{pos} as for Eq. 13. This leads to a table of 101 values of γ and θ. During the same procedure, we also calculate the *neg* versions (N_{neg}, D_{neg}, etc.) and get a second table. From these two tables, we can just get the inverted functions $\Re_{\text{pos}}$, $\Re_{\text{neg}}$ and their joint version $\Re$.

Figure 10 shows this function $\Re$ (we have chosen the *chained* strategy). The tolerance levels can be easily seen on the y-axis (on the right for each plot), which is a (linearly) normalised version of the aggregated rewards (y-axis, on the left). Apart from the convenient way of showing how the values are normalised (so all the environments become commensurable if we look at the right y-axis), we can also

[10] In the implementation, we do a trick to make things much faster: we check whether there are more than 10 consecutive 'trues'. In this case, we consider that the process has stabilised and we stop, filling the rest of values with 'true'.

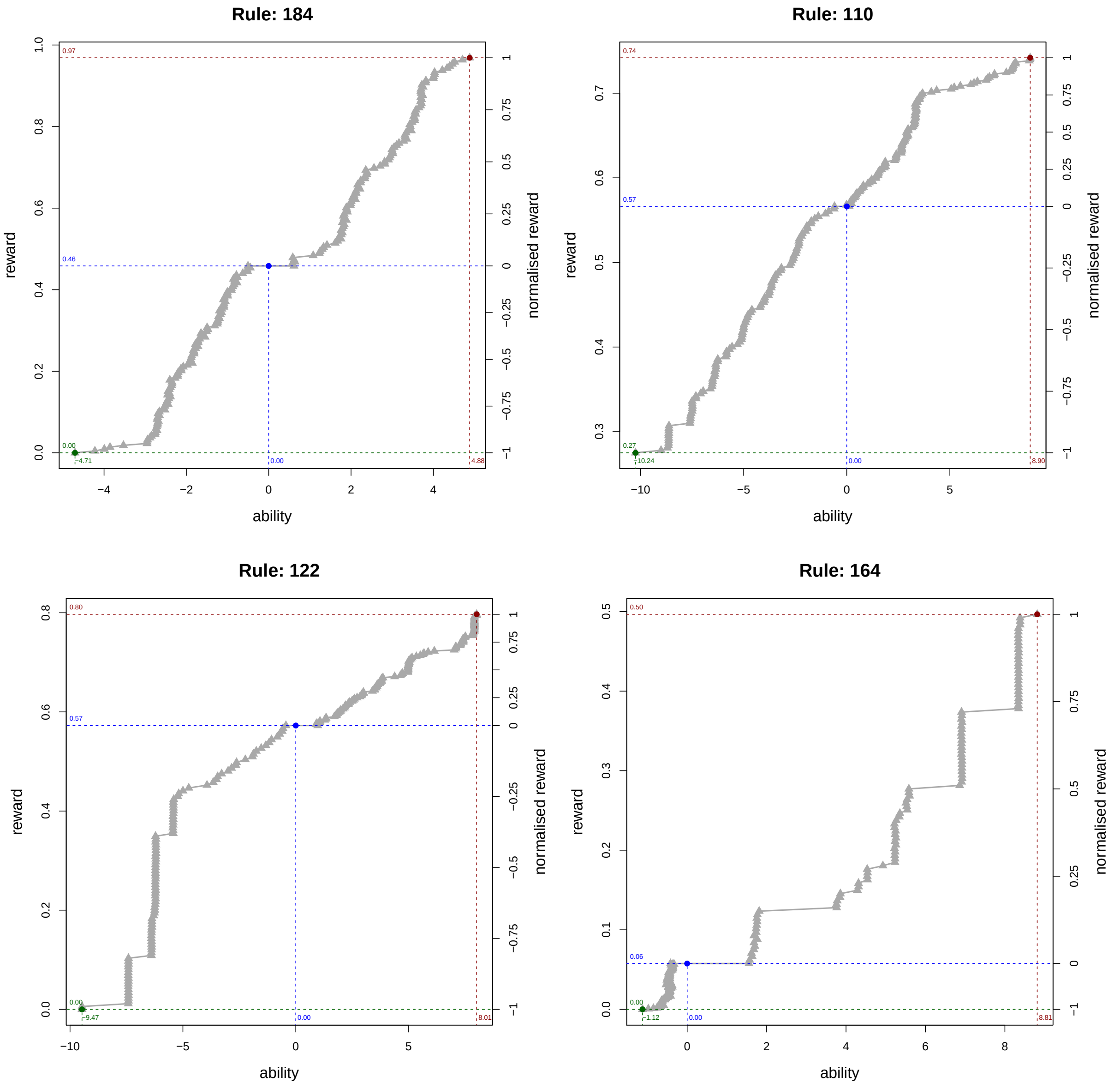


Fig. 10 Environment response curves $\Re(\theta)$ with the *chained* strategy *without* replacement for the four environments using 1,900 policies (2,000 minus the random walks), which are reduced to 1349 after removing repeated ones. The x-axis represents the ability (θ) of an agent and the y-axis (on the left) represents the expected aggregated reward R. The y-axis on the right is a normalised version, where we can use the tolerance level γ to locate any point. For instance, in order to get 1/2 probability of getting at least the maximum (0% tolerance level), we need to look at normalised 1, which requires abilities 4.88, 8.90, 8.01 and 8.81 for rules 184, 110, 122 and 164 respectively. If we set the tolerance level at $\gamma = 0.25$, we need to look at normalised reward 0.75, which gives values of 3.55, 3.10, 7.38, 8.34 for rules 184, 110, 122 and 164 respectively. More values in Table 4.

get simple and clear indicators of difficulty and discriminating power. For instance, from these curves we can get estimations of the difficulty of each environment for different levels of tolerance as calculated in the caption of Figure 10. Also, the notion of discrimination can be analysed by looking at the shape of the response curves. For instance, rules 184, 122 and 164 show curves that are below the diagonal for the positive part (top right quadrant of the curves) while 110 is above the diagonal. This indicates how difficulty varies for several values of tolerance. In fact, ability plunges from 8.90 to 3.10 when changing from 0% to 25% tolerance (normalised rewards 1 and 0.75 respectively) for rule 110. This environment (rule 110) would be a good choice if we liked to tell whether an agent has an ability higher or lower than 3, as the slope of the curve in this area is almost vertical. For higher abilities, environment with rule 164 could be used next, while environment with rule 184 would be useful to discriminate in the range 0-4, because they have high slopes in these areas. Also, it is interesting to look at these plots in a dual way,

		$D_{\text{pos}}(\gamma : 0..0.25)$					$D_{\text{neg}}(\gamma : 0..0.25)$				
Rule	$\lvert\hat{\Omega}\rvert$	0.00	0.01	0.05	0.10	0.25	0.00	0.01	0.05	0.10	0.25
184	1900 (1349)	4.88	4.72	4.33	4.01	3.55	4.73	4.20	2.94	2.76	2.53
184	9500 (5922)	5.44	5.26	4.44	4.13	3.68	4.50	4.32	3.01	2.79	2.50
110	1900 (1349)	8.90	8.88	8.07	7.74	3.10	10.24	9.01	8.64	8.63	6.97
110	9500 (5922)	12.42	12.08	10.15	10.01	8.87	11.76	11.61	10.22	9.07	8.30
122	1900 (1349)	8.01	8.01	7.91	7.91	7.38	9.47	9.47	7.42	7.41	6.39
122	9500 (5922)	8.73	8.66	8.49	8.47	7.55	10.24	9.16	8.17	8.17	8.02
164	1900 (1349)	8.81	8.38	8.34	8.34	8.34	1.11	0.95	0.75	0.67	0.56
164	9500 (5922)	10.54	10.54	9.41	9.41	9.39	1.49	1.18	0.93	0.92	0.69

Table 4 Comparison of D_{pos} and D_{neg} using the *chained* strategy for two different sizes of the working sample $\hat{\Omega}$ (in parentheses the number of policies after removing the repeated policies). Those for $|\hat{\Omega}|$ = 1,900 correspond to Figure 10 and those for $|\hat{\Omega}|$ = 9,500 correspond to Figure 11.

because just by changing the rewards r to $1 - r$ in our environments, we would get environments with the structure given by the bottom left quadrant of the curves.

Finally, Table 4 shows the differences in the estimation of difficulties using a larger working sample: 9,500 policies (10,000 without the random walks). We see that these measures are more robust and convergent, especially for tolerance levels 0.05 and 0.10. The environment response curves for the larger working sample are shown in Figure 11, and also have the same shape as those in Figure 10.

From these experiments we see that environment response curves are powerful and robust tools to analyse the difficulty and discriminating power of any environment at different levels of tolerance.

4.5 More efficient approximations

The process of inverting the functions D_{pos} and D_{neg} for the case without replacement is computationally expensive, as we discussed at the beginning of the previous section. The reason is that sampling without replacement does not lead to closed-form expressions for this inversion. One alternative is to work with the case with replacement, either because we think that it may also be a reasonable choice or just because the case with replacement is always an upper bound to the case without replacement. The process for the case with replacement is much easier. We just need to calculate the values of q as given by Eq. 9 and then combine this with Eq. 12 in proposition 2.

Figure 12 shows the case with replacement. We see that there are only small differences with respect to Figure 11. Basically the plots stretch at ability 0 and extend a little bit at the edges. So, this can be a good approximation if the computational cost of calculating the results for the case of sampling without replacement makes it infeasible for other environments.

5 Case study: a multi-agent system featuring competition and coordination

The previous case study gives us sufficient insight about how the difficulty of an agent interacting with an environment can be estimated and how the discriminating power for several environments can be used to choose some of them (or use them in a given order) when evaluating agents (as in computerised adaptive testing using IRT). However, things are more complex when many agents interact, and the difficulty of an environment is relative to the agents therein and their interactions. So it is useful to see how all these tools can be applied to a MAS setting as well. The previous case study could have easily been extended to more agents, but we have preferred to explore a different problem as a second case study to provide a more diverse set of cases.

The new experimental case study, although inherits some of the basics of the previous one, is much richer and more similar to some examples in MAS, such as the prey-predator (foraging) problem or some games such as Go. We want a setting where we can explore some configurations where cooperation is needed and some cases where competition is analysed. We also try to devise an environment that allows for complex phenomena and emergence, but we still keep its definition and policy specification

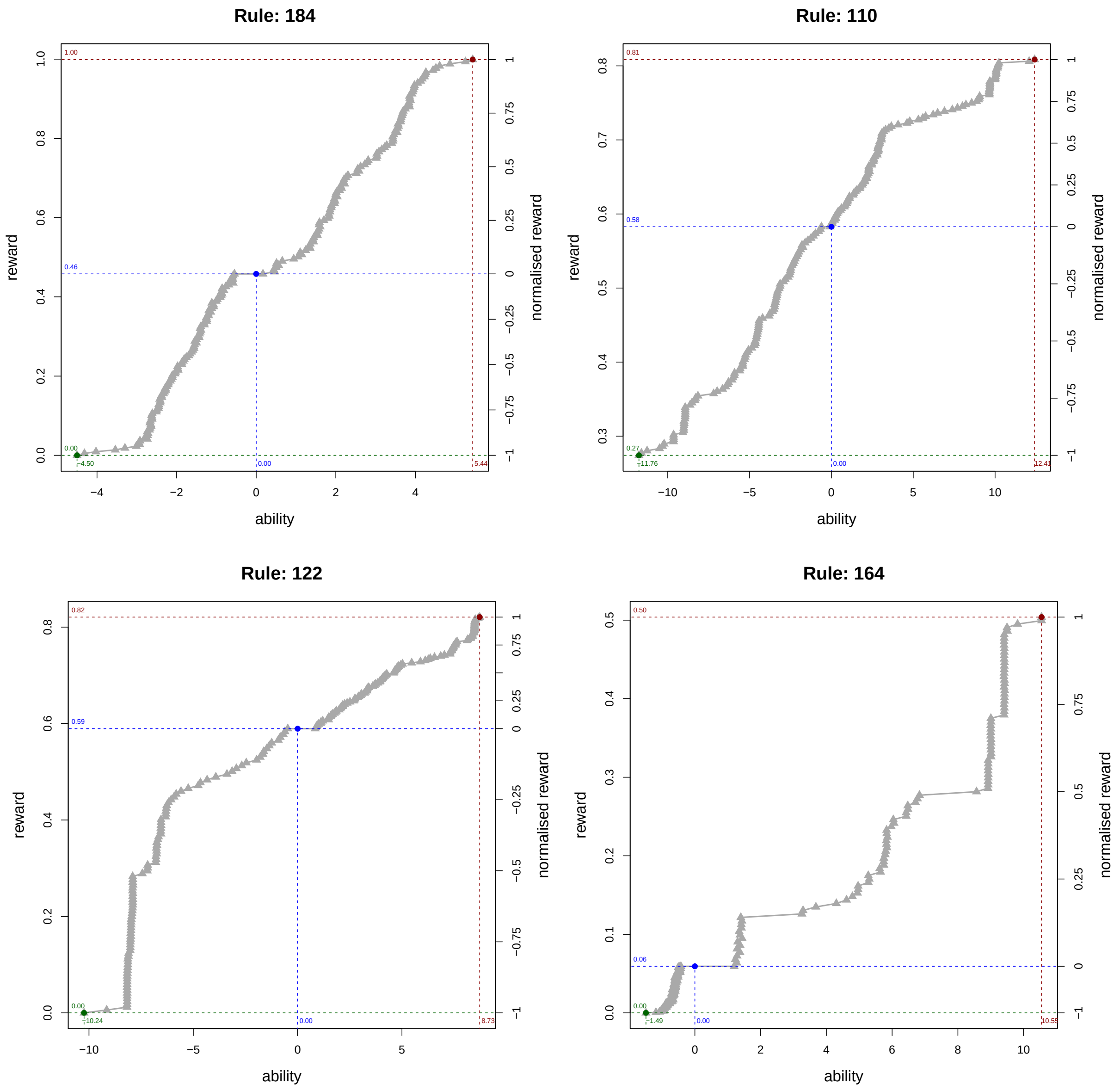


Fig. 11 Same as Figure 10 using 9,500 policies (10,000 minus random walks), which are reduced to 5,922 after removing repeated ones. This is shown for the *chained* strategy *without* replacement. The exact values of difficulty for some degrees of tolerance are shown in Table 4.

as simple as possible, without unnecessary sophistication (e.g., avoiding further dimensions, many rules, full observations, ad-hoc instructions, etc.), in order to make the following analysis shorter and more comprehensible. Nonetheless, the two case studies in this paper are just a choice and other classical problems or games could be used instead.

5.1 Surrounding and coordination MAS: definition

We would like a multi-agent problem where agents get their rewards by preying on other agents and they lose rewards when they are caught. As we want agents to behave in either way (as predators or preys), and to require the help of other agents to be able to chase another (leading to cooperation), we have just adapted the predator-prey problem in such a way that we need two predators to take a prey, by simply surrounding it. Also, we include the possibility of some collective behaviour or synchronisation by making rewards depend on a particular disposition of agents in the space. Again, we use a one-dimensional array, as it makes representation much easier through space-time diagrams, and most (if not all) complex behaviours can fit in this one-dimensional array. More formally:

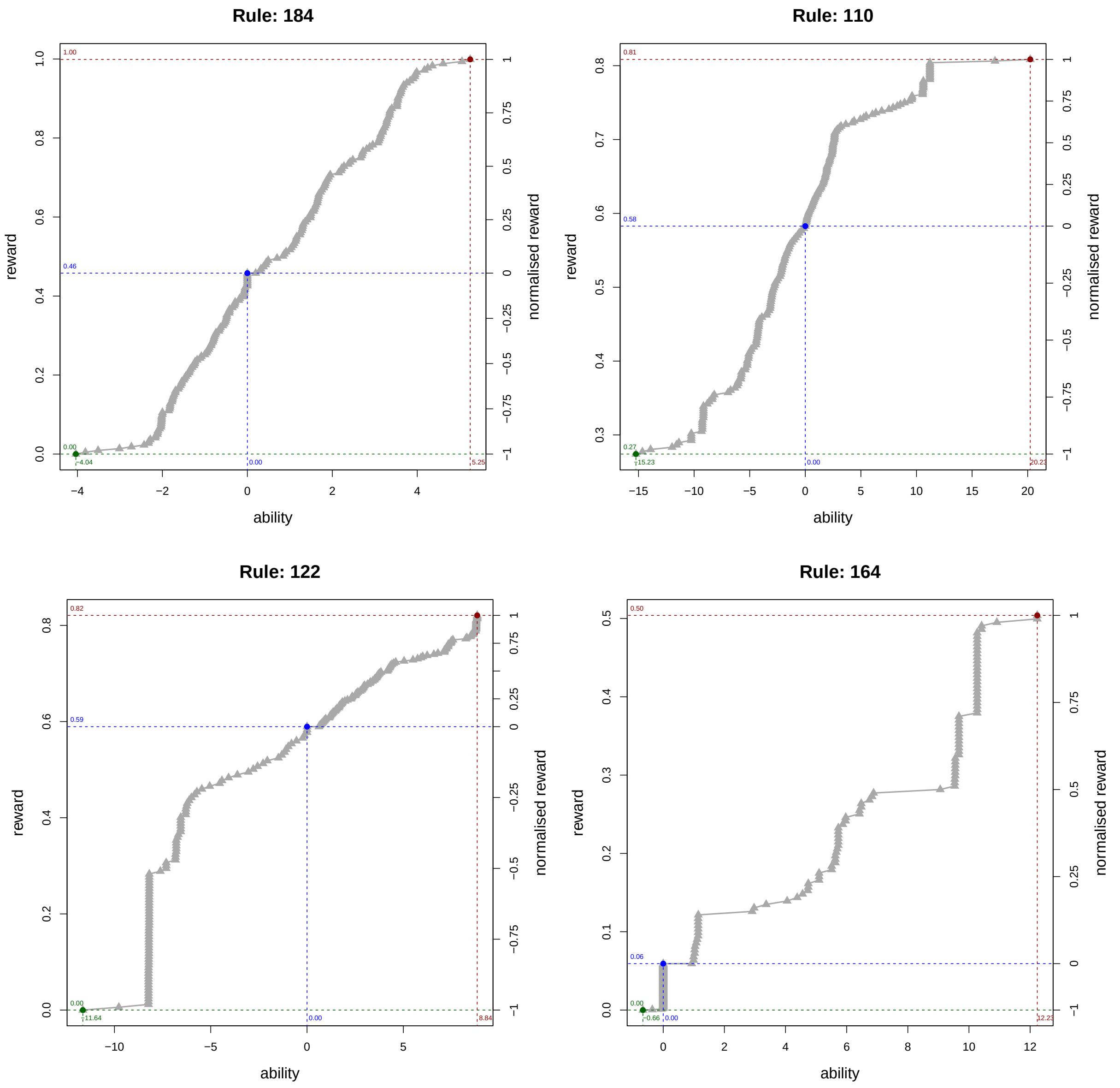


Fig. 12 Same as Figure 11 but using the *chained* strategy *with* replacement.

Definition 10 A surrounding and coordination MAS (SCMAS) is a special kind of environment $\langle \mathcal{S}, \tau, \omega, \rho, \Pi \rangle$, as seen in definition 1, with the following extra parameter $\boldsymbol{\sigma}^0$. The state space $\mathcal{S}$ is represented by a one-dimensional array of integers $\boldsymbol{\sigma} = \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle$. We consider the array to be finite ($|\boldsymbol{\sigma}| = m$) but circular in terms of neighbourhood ($\sigma_0 = \sigma_m$ and $\sigma_{m+1} = \sigma_1$). There are n agents in the environment, with $3 \leq n \leq m$. Agents occupy a cell in the array (its position p_i) with $1 \leq p_i \leq m$. Two agents cannot share a cell. There is an initial array $\boldsymbol{\sigma}^0$, also known as *seed*, whose components hold a 0 if the cell is empty or a positive integer representing an agent identifier otherwise. The set of observations $\mathcal{O}$ is given by four bits $\langle \delta_{p-2}, \delta_{p+2}, \delta_{p-1}, \delta_{p+1} \rangle$ representing the information of whether the cell two positions left σ_{p-2}, the cell two positions right σ_{p+2}, the cell one position left σ_{p-1} and the cell one position right σ_{p+1} are empty (0) or not (1). The actions $\mathcal{A}$ are given by elements in the following set { `left1`=0, `stay`=1, `right1`=2, `left2`=3, `right2`=4 }, which respectively mean to move one cell left, to stay, to move one cell right, to move two cells left, and to move two cells right.

The transition function τ is given by recalculating the new position given the actions for each agent: $\tau(\sigma^t, a_1^t, a_2^t, \ldots, a_n^t) \rightarrow \sigma^{t+1}$. This function does several checks in order to preserve the rule that two agents cannot share a cell. Basically, it is an iterative process. If the first application of the action leads to an inconsistent state (i.e., some agents share cells), then the procedure undoes the actions involved.

If this new state is still inconsistent, then the procedure undoes the actions involved. And so on until a consistent state is found. A consistent state always exists as the procedure will ultimately go back to the previous state. This transition function is deterministic, but as the agents may be non-deterministic, the whole environment may be non-deterministic.

We see that here we have a relative complex process to determine the actions that are finally executed by the system. This is implemented as a two-stage process (we get the actions, or influences, from the agents) and then we make the system react by evaluating which of them can be executed. This is similar to Ferber and Müller's model of influence and reaction [21,54]. Note that the location and concurrent actions of other agents is key for some state changes (such as two neighbouring agents swapping positions being possible if performed simultaneously).

The rewards are calculated in the following way. Rewards are zero unless an agent is in two particular situations. One situation is when there is a *surrounding* triad, i.e., three agents in a row (in neighbouring positions): $\langle ..., A, A, A, ...\rangle$. In this case, the middle agent gets reward -2 and the left and right agents get reward $+1$. If an agent is involved in more than a triad (e.g., four agents in a row), all the possible triads are applied and the rewards accumulate. Note that if an agent is in the middle of five agents in a row, this agent gets $+1+1-2=0$ as reward. The other situation involving rewards is when the agent has a *formation* triad, with a gap between the two surrounding agents and itself. In other words, we have a $\langle ..., A, 0, A, 0, A, ...\rangle$ *formation*, where A can have the index of any agent and 0 represents an empty cell. In this case, the reward of the agent in the middle is increased by 0.5. The other agents do not receive any reward by this (unless they are involved in a 'surrounding' triad or the formation continues). Intuitively, the *surrounding* triads are zero-sum rewards aimed at fostering competition (although cooperation is also possible between the surrounders), while the formation is aimed at fostering cooperation, as it can provide positive results to (almost) all agents in case all arrange in formation (agents and empty cells alternate).

The order of events for each step in the system is: observations are produced, actions are performed, the automaton is updated and finally, rewards are produced.

As we did in the previous section, we introduce a policy language, APL2, in this case:

Definition 11 The agent policy language APL2 is given by a memory (or history) binary array *mem*, initially empty (and not circular), and an ordered set of instructions $\mathcal{I} = \{$ `back`=0, `fwd`=1, `Aaddm`=2, `Aadd1`=3 $\}$. A program or policy π is a sequence of instructions $\iota_1, \iota_2, ..., \iota_{|mem|}$ in $\mathcal{I}$. The interpreter works on its memory by using two accumulators A, and the action is given by the result of the accumulator at the end of the process. Namely:

```
1.  Read the observation $\langle\delta_{p-2}, \delta_{p+2}, \delta_{p-1}, \delta_{p+1}\rangle$ and append it to the history array $mem$.
2.  Place the memory pointer $b$ at the end of $mem$.
3.  $A \leftarrow$ stay
4.  forall $\iota \in \pi$
5.    case $\iota$:
6.      back  : $b \leftarrow \max(b-1, 1)$
7.      fwd   : $b \leftarrow \min(b+1, |mem|)$
8.      Aaddm : $A \leftarrow (A + mem_b) \bmod 5$
9.      Aadd1 : $A \leftarrow (A + 1) \bmod 5$
10.   end case
11. endfor
12. return $\langle A\rangle$
```

It is easy to see that the policy 02 (i.e., the instructions `back` and `Addm`) just means the following: "if there is an agent on the left (σ_{p-1} is accessed after moving back on *mem*), then stay (action `stay`=1, as $0+\delta_{p-1} \bmod 5 = \delta_{p-1} = 1$). Otherwise, move left (action `left1`=0, as $\delta_{p-1}=0$)".

5.2 Evaluation of agent collectives

Using the MAS described above, the first thing we are going to analyse is its behaviour for a set of agents acting as a collective (or swarm). In order to do that, we will take the reward for every agent and

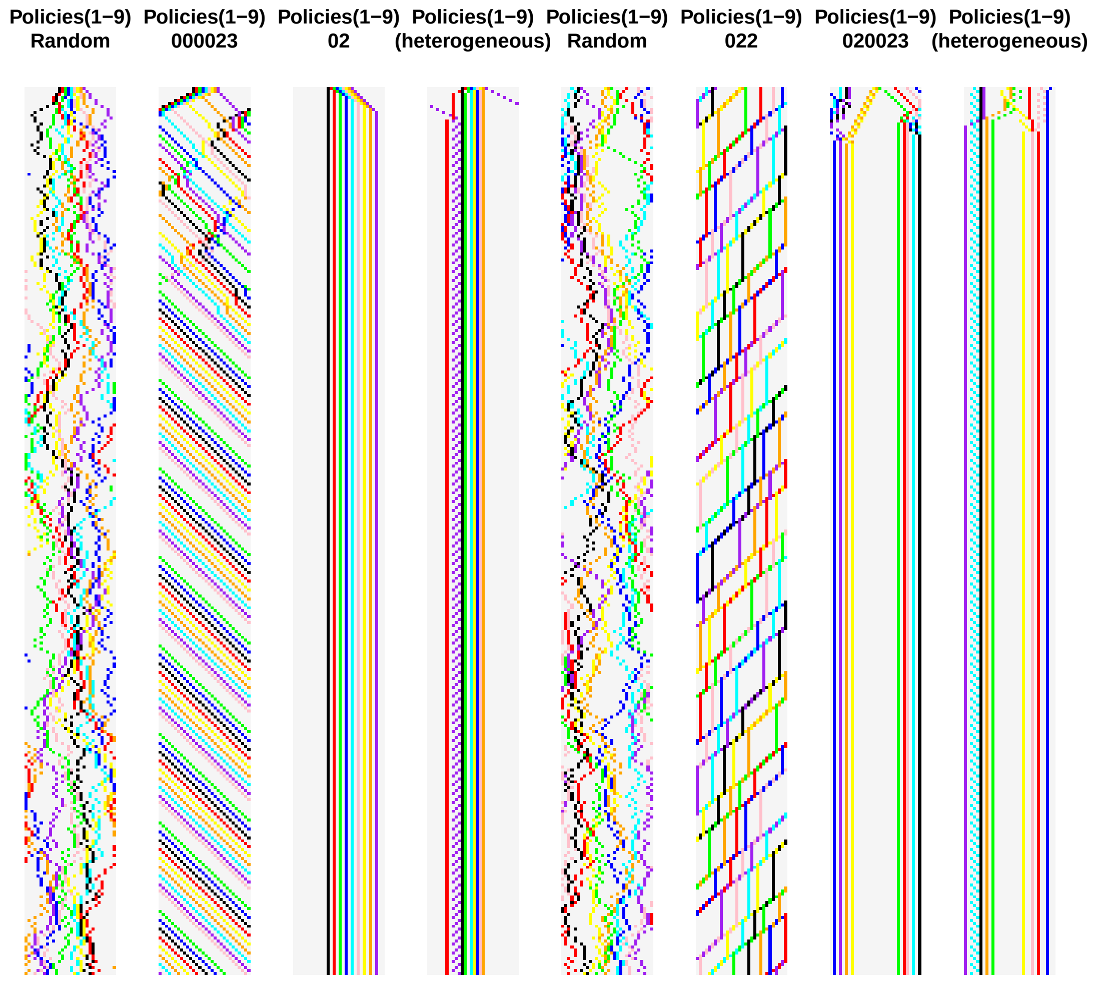


Fig. 13 Space-time diagrams (evolution for $t = 300$ steps) with different configurations for the multi-agent system SCMAS. Empty cells are shown in clear grey, and agents are shown as dots in different colours. All examples use a space of 30 cells and 9 agents. The four diagrams on the left use a start configuration where all the agents are packed between cells 12 and 20. The four diagrams on the right start with a sparser, more arbitrary configuration. Diagrams 1 and 5 show the result for random agents. Diagrams 2, 3, 6 and 7 show the results of using homogeneous configurations (all agents have the same policy). We see that diagrams 2 and 6 do not reach the *formation*, and get poor overall reward. Diagrams 3 and 7, on the contrary, converge to very good results (in fact, diagram 3 shows convergence to an optimal formation with reward close to $+0.5$). Finally, diagrams 4 and 8 show the result for a particular heterogeneous set of agents, where perfect formation is more difficult, but getting some good triads (by chance) is possible.

calculate their average. Note that, according to the reward function, 'surrounding' triads are zero-sum, so we can only get positive overall rewards for the group using the 'formation', i.e., when agents and empty cells alternate. In fact, with n agents, the maximum reward is $0.5\frac{n-2}{n}$, if the formation cannot be made circular ($n < m/2$). The minimum reward is 0. We will work with a space of 30 cells and 9 agents. Instruction programs (policies) will be generated with lengths from 0 to 20, choosing the length with a uniform distribution and then the actions also uniformly from the set of actions, as we did for the single-agent experiments in section 4.

Figure 13 shows the space-time diagrams of 8 experiments using different policies. For instance, diagrams 1 and 5 show the behaviour of the 9 agents when their policies are random (they choose an action randomly). As expected, the behaviour is chaotic and the expected reward is low. We are more interested in other configurations where (all) agents do not behave randomly. We will first explore the case of a *homogeneous* MAS, by assuming that all policies are the same (note that this does not mean that the actions are the same). In other words, homogeneous means that we choose a policy for agent 1, and this policy is also used for agents 2 to 9.

Now the question is how difficult it is and how much diverse the results are for a homogeneous set of agents to arrange in a 'formation'. We must analyse each problem *instance* separately, as we did with the single agent case in section 4 using a fixed seed for all the iterations. This is important, as we do not want to analyse how difficult it is to find a good starting configuration (e.g., the set of agents might already be in formation and a very simple 'stay' policy for all would suffice). Instead, we want to know how difficult it is to find a policy for the MAS for a given starting configuration. This also supports the idea of using a *fresh* strategy, which seems more natural in this case. For instance, diagrams 2 and 3 in Figure 13 show the very different results of two policies with a starting configuration where all agents are consecutively arranged initially. Diagrams 6 and 7 show results for other policies with a different starting configuration.

We can focus on the diversity of policy rewards depending on the starting configuration. This is what we do in Figure 14 (top). The plot on the top left shows that with a regular start configuration, it is easy to find policies achieving about 0.38 mean reward (where the maximum is $0.5\frac{7}{9} = 0.389$). However, this seems to be more difficult for a less regular starting point, as shown in the plot on the top right of Figure 14. This can be seen more clearly in Figure 15 (top), as we can compare several cases with a standardised measure of ability. The plot on the top left shows that abilities around 3 are enough for this regular starting point, while things become more difficult with a non-regular start. Also note the difference in discriminating power. The seed on the top left would be useful to tell between abilities below and above 3 only (but very sharply), while the seed on the top right can be useful to distinguish (possibly less robustly because of its coverage) a wider range of abilities.

We can use our tools and plots to analyse the *heterogeneous* case as well. Here, the nine agents may have different policies. We generate their policies as usual, just for each iteration. As we evaluate how well the nine policies work, a search strategy would not discard a policy with a bad result, because it may depend on the others. For this reason, the response curves can be calculated with replacement. As we can see in Figure 14 (bottom), the results change significantly in comparison to those of a homogeneous MAS. First, in both bottom-left and bottom-right, we do not see cases with an average reward greater than 0.2. So, solving these two instances with heterogeneous MAS is more difficult than with a homogeneous one. This can be seen more clearly in Figure 15. In fact, with the response curves, we can see that the irregular start configuration (bottom right) is better than the regular one (bottom left). This is disguised in the distribution plots of Figure 14, as the bottom-left plot is very narrow and only raised by an isolated high point at complexity 8.

If we take a global look at Figure 14, we see how different the results are with respect to the use of random policies (designated with 'rnd' on the complexity x-axis). Note that random policies have a very small deviation and do not say anything about the difficulty of the problem. This again justifies the use of a random population of policies instead of assuming a random behaviour. Overall, we see that figures 14 and 15 give us very valuable and precise information for the multi-agent case also, either with homogeneous or heterogeneous sets of policies.

5.3 Evaluation of individuals

A MAS can be analysed from the point of view of the performance of the group as we have done above, but it can also be analysed from the performance of each individual. This leads to competition and other forms of cooperation. With our system SCMAS, it is easy to focus on individuals by considering rewards separately. The maximum reward for an individual is $+1$ (if it surrounds other agents all the time) and the minimum reward is -2 (if it is surrounded indefinitely). Initially, we will use 9 agents (with a heterogeneous set of policies) and a similar configuration as in section 5.2. We can see some examples in Figure 16. For instance, the first space-time diagram shows the result of a fixed policy (agent #1, in black) against 8 agents behaving randomly. As we can see, it is very difficult for agent #1 to surround another agent for some time. Diagrams 2, 3 and 4 use a fixed heterogeneous set of policies for agents 2-9 and show very different results. Diagrams 2 and 4 show policies that behave very well, and stabilise with an optimal reward for #1, while diagram 3 shows a case where agent #1 is captured and its policy does not allow it to escape.

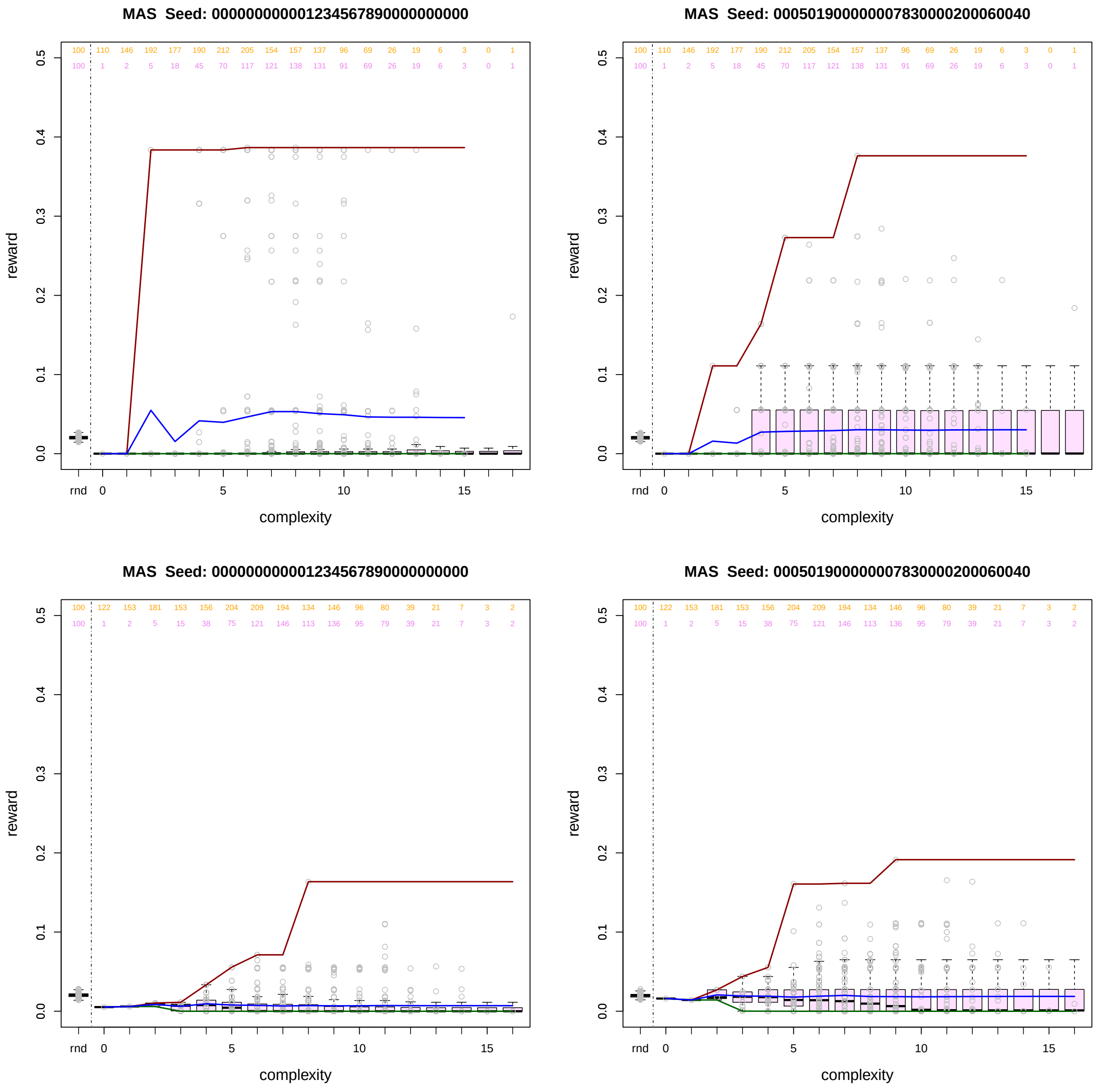


Fig. 14 We show the distributions $R[k]$ for different configurations for the multi-agent system SCMAS. All examples use a space of 30 cells and 9 agents. We generate 2,000 agents (100 of them are random) and for each of them the environment runs during 300 iterations ($t = 300$), starting from scratch for each agent (*fresh* strategy), using the different seed configurations in each case, as shown. The two plots on the top use a homogeneous policy (the same policy for the 9 agents), while the two plots on the bottom use a heterogeneous policy (different policies for each agent). We see that the use of a homogeneous policy is better for both configurations, as it is easy to obtain a 'formation' arrangement if all agents have the same behaviour, especially for the seed where the agents start as a packed row.

One might think that a fair way of evaluating an agent against some other agents in a MAS environment is to make it compete with random rivals. However, this choice is not appropriate, as we can see on Figure 17 (top left), as it leads to poor results and poor variability (with very low discriminating power on Figure 18, top left). As an alternative, we can focus on a fixed set of rivals and a fixed initial configuration. A set of agents was generated randomly (just once): π_2 = "220023", π_3 = "00022333", π_4 = "120200233", π_5 = "22", π_6 = "21112333", π_7 = "2233", π_8 = "1123", π_9 = "". As we can see on Figure 17 (top right), this gives a richer perspective, with maximum and minimum rewards achievable. However, this represents just a particular case, which can be useful to analyse the ability that is required to get good results given a particular set of opponents. This is more evident in Figure 18 (top), where we see the difficulty of both cases.

The comparison of particular cases is interesting, but we would like to know the ability of an agent against varying sets of agents. For this, we cannot treat each experiment separately, as this will, again,

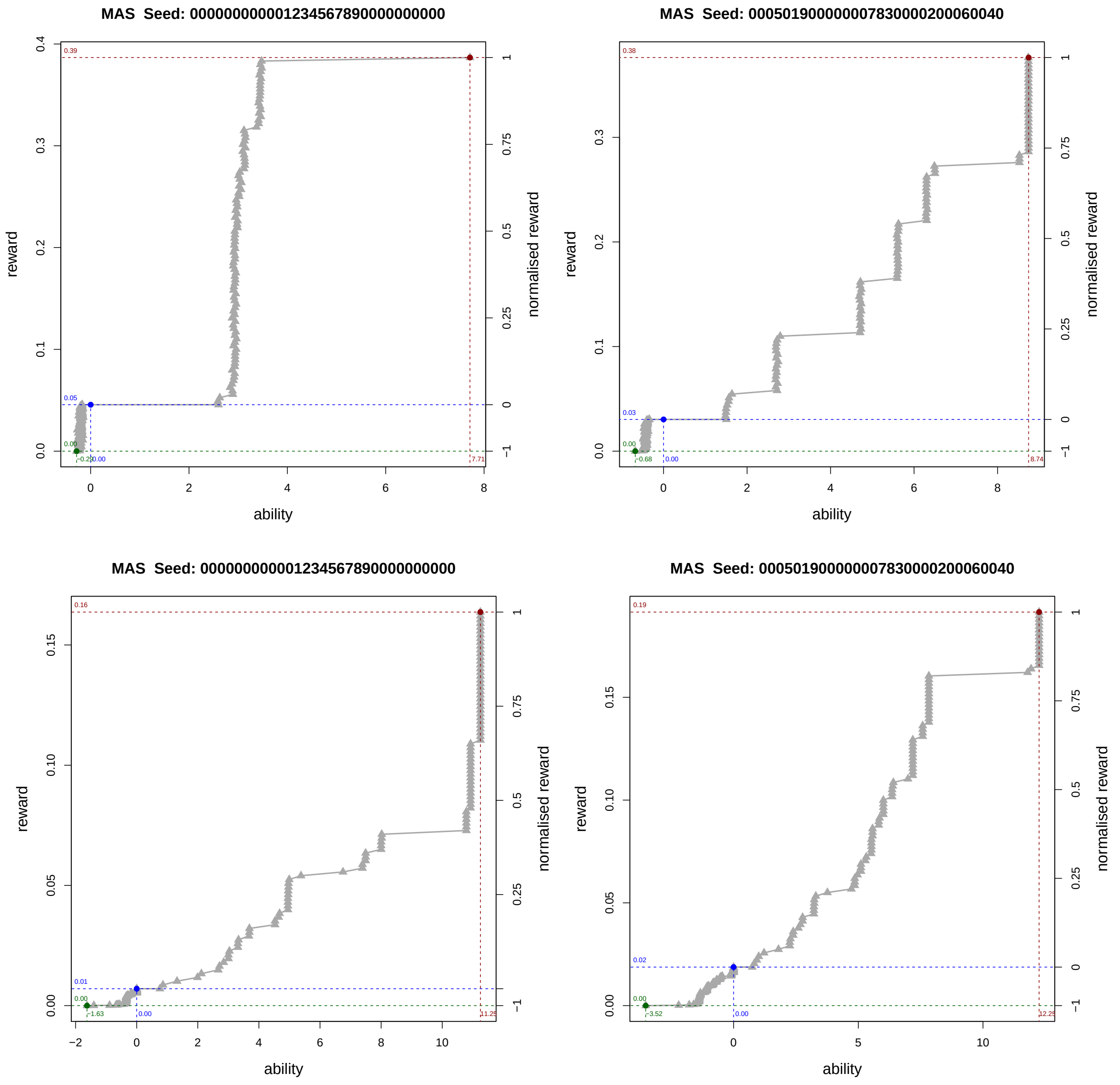


Fig. 15 Environment response curves $\Re(\theta)$ with the *fresh* strategy for the four environments using 1,900 policies (2,000 minus the random walks), corresponding to Figure 14. The difficulties of the homogeneous policies (top) are calculated without replacement while the difficulties of the heterogeneous policies (bottom) are calculated with replacement. We now see very neatly that the homogeneous policies with a regular seed (top-left) lead to very easy situations and highly discriminating power around ability $\theta = 3$. The bottom plots (heterogeneous case) also show that abilities around 7-10 are needed to get a reward of 0.15, while the plots on the top reach this value with abilities around 3-4.

show the chance that an easy set of opponents could appear. Instead, each experiment needs to calculate the expected reward of a policy π against a distribution of population sets. This can be modelled quite easily by fixing one policy and repeating the experiment against many sets of opponents (using the same distribution we use to generate policies) and *average* the reward results. In our case, we use 50 repetitions, which gives a reasonably good estimation in this case. The results are shown in Figure 17 (bottom left). We see that it is very difficult to find a policy which is optimal against any set of opponents. The best that can be found is an expected reward of 0.31 at an ability of almost 9, as we can see in Figure 18 (bottom left).

Our last plot in these figures represents an interesting combination. From the previous experiment, we select the best 8 policies, and use them for the opponents. Namely, the best policies are: π_2 = "20222333", π_3 = "2202122000233", π_4 = "0212202112233", π_5 = "022120211223", π_6 = "112223", π_7 = "022000022333", π_8 = "200022333", π_9 = "12112202120233". These policies obtained values between 0.32 and 0.26 in the previous experiment. Now, we repeat the experiment using these policies,

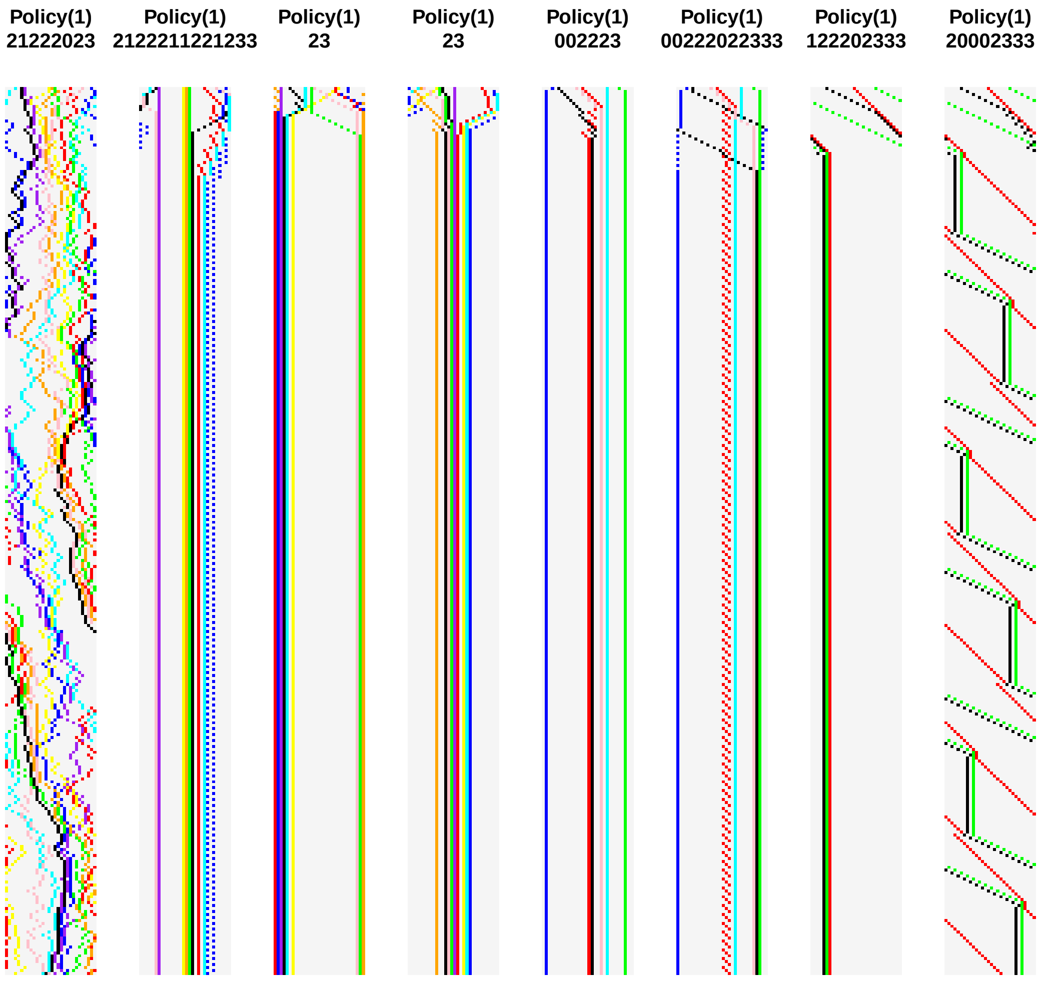


Fig. 16 Space-time diagrams (evolution for $t = 300$ steps) with different configurations for the multi-agent system SCMAS. Empty cells are shown in clear grey, and agents are shown as dots in different colours. We focus on the rewards of just one individual, agent #1, which is shown in black. All examples use a space of 30 cells. The four diagrams on the left use 9 agents, diagrams 5 and 6 use 6 agents and diagrams 7 and 8 use 3 agents.

as trying to simulate a case where an agent has to compete against the best. The results, shown in Figure 17 (bottom right), are slightly surprising. While the mean reward (in blue) is negative, which is consistent with a situation where the agent competes against very good agents, the maximum and minimum achievable rewards are widened. In fact, there are even a few cases where a reward of 0.43 can be obtained against this set of best opponents (see the bottom-left plot of figures 18 and 17). Several explanations can be given here, but we think that some of these competing agents will try to create triads, which can also be exploited by a few shrewd policies, as in this case.

Finally, the number of agents could be subject to study as well. We have performed several experiments changing the number of agents. We have observed that the difficulty of getting high rewards seems to increase very slightly as the number of agents decreases. This may be explained in this case as the possibility of creating triads is reduced with fewer agents. Of course, for numbers of agents approaching the number of cells, difficulty also increases as it becomes too crowded and mobility is reduced (note that agents cannot share cells).

Overall, the experiments performed in this section have shown the usefulness of the distribution plots and the response curves to analyse the difficulty and discriminating power of several situations: homogeneous multi-agent groups (or swarms), heterogeneous groups, competing individuals against a sample of agents and competing individuals against the best opponents.

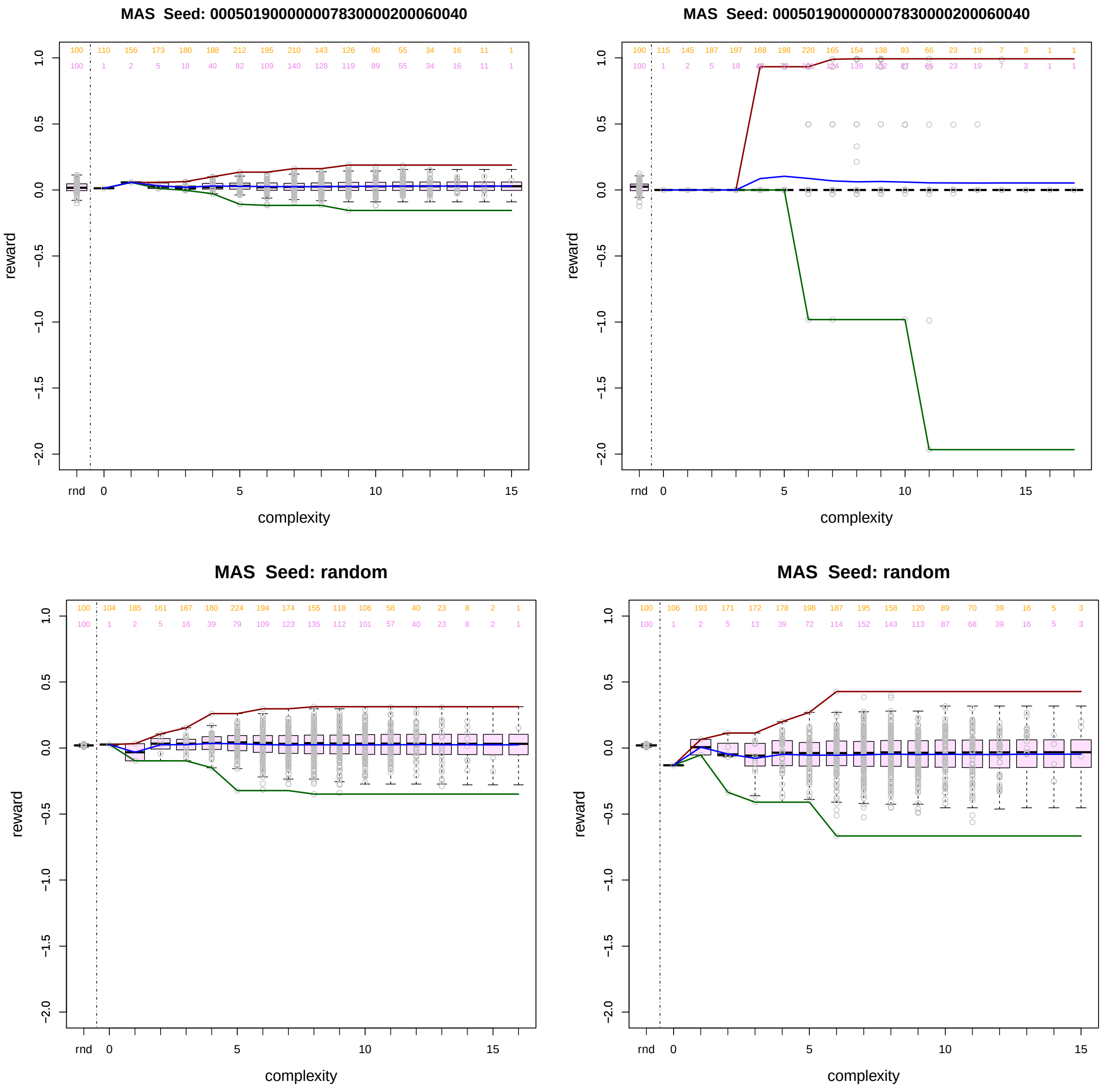


Fig. 17 We show the distributions $R[k]$ for different configurations for the multi-agent system SCMAS. All examples use a space of 30 cells and 9 agents. We generate 2,000 agents (100 of them are random) and for each of them the environment runs during 300 iterations ($t = 300$), starting from scratch for each agent (*fresh* strategy). The two plots on the top use different seed configurations in each case, as shown. The top-left plot examines what happens when an agent has to compete with 8 random agents. As we can see, the results are poor in terms of rewards and variability. The top-right plot examines what happens when we fix the other 8 policies (and keep them for all the experiments). For this choice of competitors, there seems to be easy policies for #1 to succeed. As we see that the results highly depend on the seed and the other agents, the bottom plots show the results when we confront each policy with 50 different teams of competitors (with different seeds also). This means that we perform $2,000 \times 50 = 100,000$ experiments (300 iterations each). This is what we see on the bottom-left plot. Finally, the bottom-right plot shows the results when we choose the best 8 agents from the previous experiment and make them be the competitors of agent #1 with varying seed configurations. Surprisingly, it seems easier to compete against the best to get good rewards than against a random choice of policies (but note that the average reward is lower).

6 Discussion

In section 2 we mentioned many other approaches to the notions of difficulty and discriminating power. We will now see the relation between our proposal and other notions of complexity, and we will analyse the use of more sophisticated policies in the context of multi-agent systems.

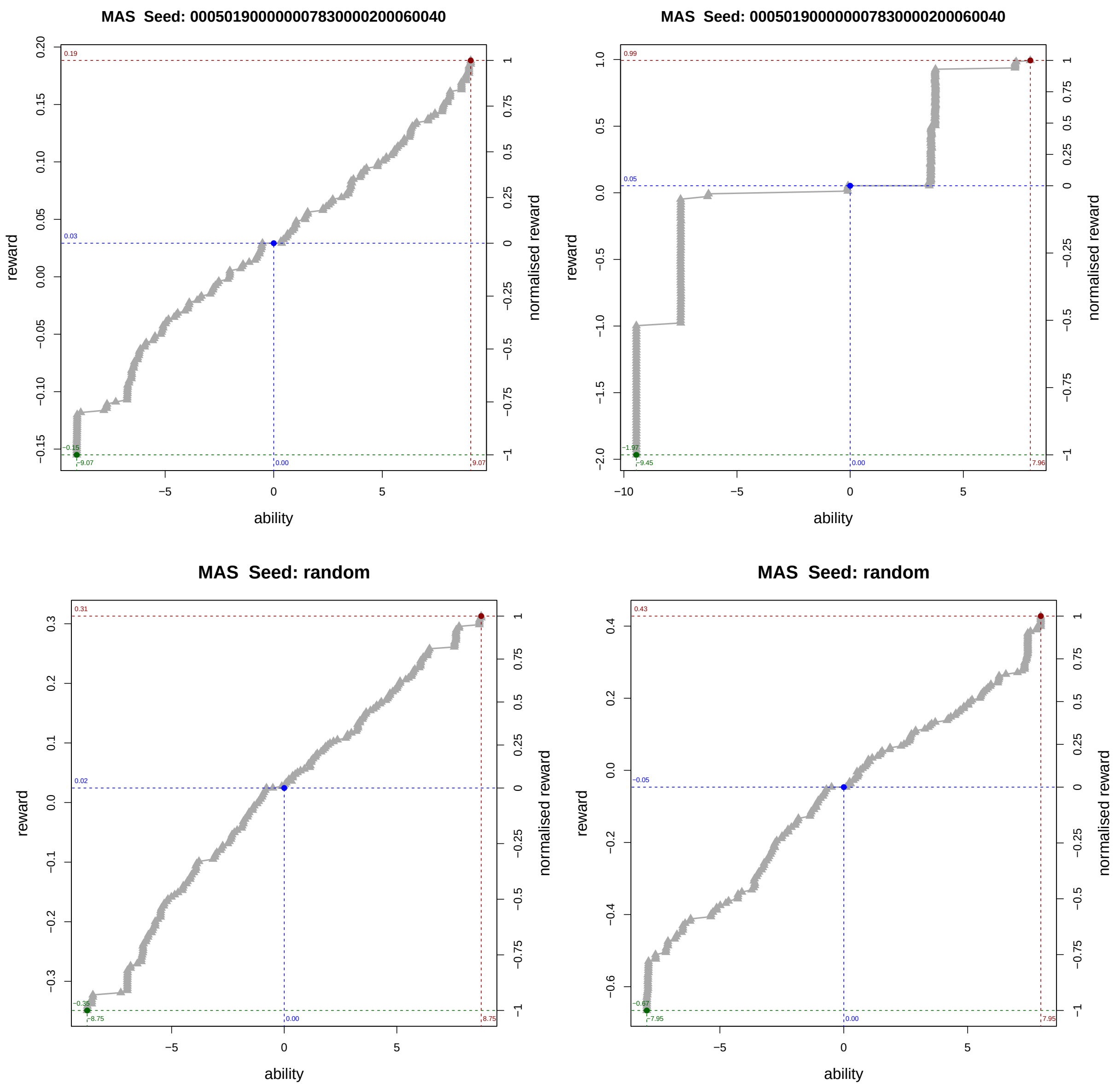


Fig. 18 Environment response curves $\Re(\theta)$ with the *fresh* strategy for the four environments using 1,900 policies (2,000 minus the random walks), corresponding to Figure 17. The difficulties are all calculated without replacement, as the opponents are fixed for the four cases. We see that an ability of 5 only reaches expected reward 0.10 for the top-left case (competing against a set of random agents). The specific case posed by a given set of opponents (shown on the top-right plot) is very easy and highly discriminative for low abilities. The bottom-left curve shows the results using each policy against 50 teams of opponents and averaging rewards as if it were a single experiment. We see a wide range of abilities and a relatively poor maximum result (0.31). We can see that the bottom right curve has a negative expected reward at ability 0, which means that competing against good agents leads to bad results for unable agents. However, we see that the best results for higher abilities are even higher than the bottom-left case.

6.1 Difficulty and complexity

In section 2 we discussed that it can be argued that upper bounds of the difficulty of an environment can be estimated by evaluating the complexity of the whole environment. In fact, in the setting seen in section 4, this would have led to the size of a compressed representation of the definition of the configuration rules, the reward rules and the observation function; a large number. For the multi-agent environment in section 5, this would have been clearly unfeasible. For instance, the implementation of the environments and automata of the first case study takes about hundreds lines of code in the language R. More compressed implementations are possible, but it is difficult to conceive an implementation whose length is less than a few thousand bits. This would be a very loose upper bound for complexity (and hence meaningless) taking into account that some problems are very easy.

A way out might have been to consider some underlying setting or class of environments, from which we would just add some parameters or rules. For instance, we could assume that the reference machine contains the basics for evolutionary cellular automata as well as the way agents and rewards are handled in this setting. With this assumption, we would only need to calculate how complex each rule is. The problem here —and the use of elementary cellular automata makes it emphatically clear— is that there are only 256 rules with the same information content each: $\log_2 256 = 8$ bits, and each of them leads to very different behaviours and difficulty values.

Another possibility might have been to look at the patterns each ECA rule generates and see whether this correlates with the difficulty we can find in them. Actually, these patterns show high variability. In fact, a deep and insightful analysis of the space-time diagrams has recently been performed by several studies [63,65]. Note that if we analyse and try to compress their space-time diagrams (e.g., those depicted in Figure 4) we are looking at some of their emergence properties, not at the automaton itself. In other words, we only code its emergent properties, and not the whole of it. As a result, the complexity may be much lower than the whole automaton. If we just focus on the four rules we have used throughout the experimental section: 184, 110, 122 and 164, we have that Zenil's approach based on compression [63], sorts them (from lowest to highest complexity) as follows: 164, 184, 122, 110. A related study, [65], which uses the coding theorem method through a 'block matrix decomposition', reaches slightly different values but exactly the same order. If we compare this order with the results of difficulty (with 0 tolerance) as shown in Figure 10, we have that the order is 184, 122, 164 and 110, and for (0.25 tolerance) it is 110, 184, 122, 164. In other words, the Kolmogorov complexity of how the ECA evolves in space-time does not seem to correlate with the difficulty that the agent may find in it. This is of course expected, since there are many other things (such as rewards or the influence of the agent in the ECA) that may lead to simple policies occasionally succeeding even if the configuration of the automaton becomes too messy (complex). This is even clearer in the multi-agent case, as the characteristics of the other agents (even simple ones) have very important effects on difficulty.

The take-away message of all this is something that we already pointed out in the introduction: the emergence of complexity is very different to the emergence of difficulty. While some approaches to difficulty can work (and have worked) by looking at the environment complexity, any general account of difficulty must take an *agent-centred* point of view.

This does not mean at all that Kolmogorov complexity and other tools from algorithmic information theory should not be applied for determining the difficulty of an environment. The lesson learnt is that in order to understand difficulty we need to focus on the policies, as we have done in this paper. In fact, the good thing about Kolmogorov complexity applied to policies is that the results should be more independent of the language used to represent policies (our agent policy languages APL and APL2 here). Obviously, there may be significant differences if the language uses some higher-level features, such as AgentSpeak [7], 2APL [13] or other BDI approaches [51], and the calculation may be more difficult, but the main principles would still apply.

6.2 Policies in multi-agent systems for cooperation, competition and communication

In the two case studies that have been analysed in the previous sections, we have worked with very concrete actions and very simple observations. The policy language is able to combine these two things, using a memory and some accumulators in order to show complex behaviour, including conditions and actions that depend on current and past observations. Nonetheless, actions, environments and policies can be described in terms of more higher-level concepts such as intentions, emotions, etc. This means that knowledge-level agents could also be considered, especially those that can express beliefs, desires and intentions, and can do some form of communication. There is nothing against this, apart from finding a compact way of expressing the policies.

The existence of other agents make the mere accomplishment of a written policy much more cumbersome. For instance, in our case studies a policy may be described in basic actions such as move right, but occasionally the action cannot be performed because there is an agent blocking the cell. If this action could be attempted in subsequent iterations (perhaps through a 're-attempt' action) policies could become simpler. Also, more abstract policies could be used, as well as explicit separations between intentions and actions (as in the influence-reaction model [21,54]). For instance, a policy where basic actions could be 'approach the rightmost agent' would be more powerful in terms of knowledge, interaction, intentions, situation and awareness about the other agents. This is acceptable as far as all agents we want to compare use the same representation language for policies. Otherwise, the comparison would not be fair.

As a result, these techniques could be applied to common AAMAS scenarios, such as several kinds of predator-prey (foraging) problem or games (such as pacman), planning problems (such as the blocks world), navigation problems, traffic or pedestrian simulation, swarms, zero-sum and non-zero-sum games, robotic scenarios, and many other areas. Response curves, for instance, can be very useful because many environments (or the same environment with several parameters, opponents or initial conditions) can be compared with a standardised scale of abilities. It may be difficult to apply this to some problems used in competitions (e.g., Robocup) as many participants will have very complex programs and their policies could not be compared to other (basic) policies. In competitions, it would be necessary that all agents share the same specification language, so policies are comparable. For instance, agent specification languages, such as AgentSpeak in Jason [7] (or others, [8,47,2,13]) could be used to express several agents for a given problem. For instance, we could think of the gold miners example [7], where two teams of miners need to coordinate in order to collect and deliver as much as gold as possible. We would need to specify the rewards that are shared or assigned individually, as well as whether we measure the complexity of the policy of one agent or a set of agents, and whether we analyse homogeneous or heterogeneous teams (as we did with the MAS case study in this paper).

The most complex part would be the generation of policies. One problem about AgentSpeak is that random strings may lead to syntactically (or semantically) incorrect programs. So the idea would be to use operators in such a way that new instructions can be added and removed from the program, while keeping the program consistent. Also, it would be much easier if the program is constructed with a skeleton including the most critical parts, and the 'logic' of the policy is left open to perform modifications on it. Adapting the techniques in this paper to a problem such as this using AgentSpeak would be a research work on its own. However, for other kinds of classical problems in agent theory or game theory, especially when the number of actions is reduced and we want to analyse equilibria, deadlocks, etc., the adaptation might be much easier. For instance, another example would be the application to RL-competitions [59], where the environments, actions and interface are well specified beforehand, and tools such as RL-glue can be used to determine a standard policy language for all.

One thing that we have not had space to fully analyse is the relation between cooperation and competition in the MAS case study. An agent can be good in a competitive situation but bad in a cooperative one, and all this can change depending on the team-mates or the opponents, as we have seen in the MAS case study. The use of the ability scales to construct better rankings for populations of agents could be useful for contests, especially those where there are matches (one vs one games). The use of response curves and opponent-dependent abilities would be novel in the context of IRT and computerised adaptive testing.

Finally, as we mentioned in section 3.1, we are assuming certain features about how problem-solving systems generate and test their policies, using different types of search strategies. In a multi-agent system with communication, some team-mates could be useful for exploring the problem space and sharing results with the rest, so updating the policies that need (and need not) be explored. Also, if agents can learn from other agents in a multi-agent system, as they see the effects of some policies performed by other agents, then this could suggest that the sampling of the population as has been done in this paper would need to be overhauled.

7 Application procedure

In sections 4 and 5 we saw the application of our proposal to two different kinds of environments, with slightly different, but simple, languages. In the previous section we have discussed about how crucial the choice of expressive (and more abstract) policy languages can be for some multi-agent systems, and the need of deriving measures and approximations of (Kolmogorov) complexity for these languages. While this choice is most important, it is useful to have an overall view of the whole process, and see that many other steps will be easier and systematic. Also, in the following section we will discuss several applications. In order to facilitate them, we will summarise a procedure, as a series of steps, that has to be performed whenever we want to apply the approach that has been introduced in this paper. Namely, if we want to give a characterisation of the difficulty and discriminating power of a given interactive task, we need to follow the steps shown in Figure 19, and enumerated below:

1. Identify the environment μ that represents the task, its interface and the agent roles.
2. Define or choose an agent policy language L that is simple but powerful such that it can represent diverse agent behaviours.
3. Define a measure of complexity for the policies, $K(\pi)$. If meaningful programs in the language L are large, this measure should focus on the parts of the policy programs that are more closely related to agent behaviour.
4. Derive an a priori distribution from the previous complexity, e.g., normalising $w(\pi) = 2^{-K(\pi)}$.
5. Make a sample Ω of policies using w, but ensure that a wide range of complexities is covered. Determine how this distribution is going to be sampled (with or without replacement) according to the search strategies (e.g., Monte Carlo, Levin's search, etc.) that are more reasonable for the given task represented by the environment.
6. If it is a multi-agent system, choose or sample the rest of agents (from the same or a different distribution).
7. Execute all the $\pi \in \Omega$ in μ for a sufficient number of steps. If there are stochastic components or we want results to be valid for several initial conditions, parameters or competitors, repeat the experiments and average the aggregated rewards $R(\pi)$ for each policy π and configuration.
8. Draw a scatter plot of $R[k]$ for all policies, i.e., the complexity $K(\pi)$ on the x-axis and the aggregated reward $R(\pi)$ on the y-axis. Derive some statistics from it: maximum, minimum, average, etc.
9. Derive the environment response function from Ω and w using definitions 4, 5, 6 and 7 and equations 10, 11, 13 and 14. If there is not an important reason for using the formulas and procedures without replacement, use the formulas with replacement, as they are simpler.
10. Plot the environment response curves. Find the range of abilities (values on the x-axis) where slopes are high so the environment can discriminate for those areas.

Note that if the language L, complexity $K(\mu)$ and distribution $w(\pi)$ are not biassed in favour or against positive or negative results, the scatter plots, the indicators and the environment response curves can be used (exchanging *pos* and *neg*) if the rewards are just negated.

The evaluation of a single environment is useful to understand how different agents behave, to see whether their aggregated rewards are comparably low or high, as well as determining whether the policy language is meaningful and powerful enough to represent rich and successful policies. However, much more interesting things are possible if we do this procedure for several (parametrised) environments. In this case, we can keep a collection of pre-evaluated environments and use them to evaluate agents by the use of adaptive testing techniques, using the environments in an appropriate order. If the environments have been evaluated with the same policy language, the results will be comparable, and we will be able to determine which environments are more or less challenging for that policy language. Interestingly, if we change the policy language and re-evaluate the environments, we can see whether the new policy language leads to lower or higher difficulties. This can help find more suitable (and possibly more abstract) agent policy languages for some environment domains.

8 Conclusions

The main contribution of this paper is the view of difficulty in terms of how a population of agent policies (i.e., behaviours) performs on a given task. While this is a traditional view in psychometrics, especially in Item Response Theory, here we can apply this view to non-biological subjects, such as AI agents. For this purpose, we construct a population of agents (or behaviours) according to some distribution over a policy language, and evaluate their results in terms of the (Kolmogorov) complexity of the policy. The

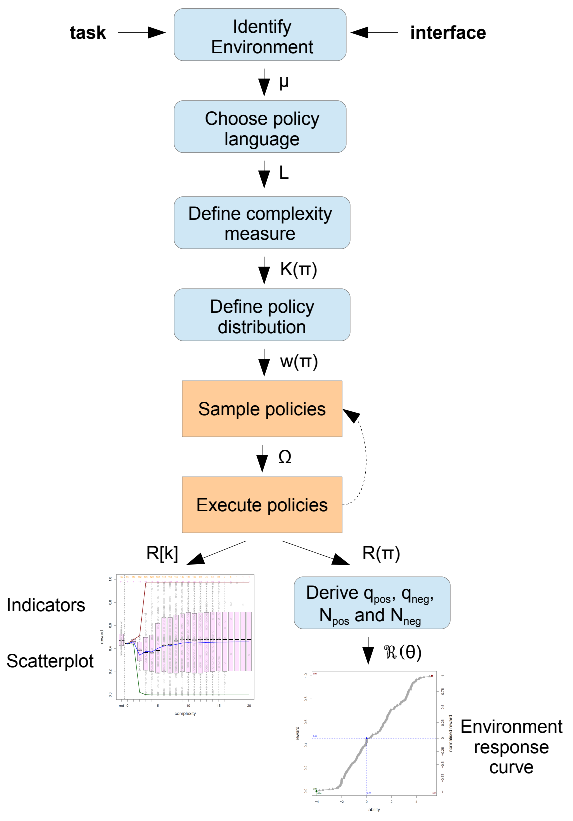


Fig. 19 Workflow for the procedure that is suggested to evaluate the difficulty and discriminating power of a given interactive task (and interface), represented by an environment. The result of the process is a series of indicators and plots that provide a detailed characterisation of the ranges of abilities that the environment is able to cover and discriminate.

development of these ideas yielded several findings. The distribution of policies in terms of complexity, either by the use of graphical tools or statistical indicators is informative. However, these plots and indicators are sometimes hard to understand and do not yield clear notions of difficulty and discriminating power, nor how several environments can be normalised so their results become commensurable. The development of the environment response curves gives a much cleaner picture of the notions of difficulty and discriminating power and makes the analysis directly operative for constructing tests and benchmarks, as Item Response Theory does. The distribution of policies in terms of complexity, either by the use of graphical tools or statistical indicators is informative. However, these plots and indicators are sometimes hard to understand and do not yield clear notions of difficulty and discriminating power, nor how several environments can be normalised so their results become commensurable. The development of the environment response curves gives a much cleaner picture of the notions of difficulty and discriminating power and makes the analysis directly operative for constructing tests and benchmarks, as Item Response Theory does.

The first application of the tools introduced in this paper is the one that motivated it: the construction of cognitive tests of different abilities (for artificial or natural systems) or the development of better competitions or evaluation benchmarks, in the context of universal psychometrics [36]. With the tools developed here we can now choose an environment class representing a cognitive ability, and select a subset of environments to integrate a test. This selection can now be done in a much more principled way by using the results of a population of policies or agents. First, we can normalise the results of many environments using the magnitude of the ability and a normalised reward, so the aggregation of results of several environments (tasks) becomes more meaningful. Second, we know their difficulty at any tolerance level, so we can choose those tasks that are more appropriate to the agent we want to evaluate. Third, by using their discriminating power, we can get maximum information from a single environment, and hence accelerate the evaluation. In fact, we can adapt techniques from (computerised) adaptive testing to do this. For instance, if we want to evaluate an agent for the MAS case study, we could use the environment on the top right of Figure 18, to know whether the agent's ability is higher or lower than 4, as this environment is very discriminative around this point. From here, we could choose other environments whose response curve characterisation focusses on lower or upper ranges of the ability axis, depending on the result of the previous environment. With just a few environments, we could have a good approximation of the agent ability (instead of dozens of evaluations that would be required to have a reliable estimation with environments that have not been characterised previously). The generality of this approach makes it applicable to any kind of agent or system since environments are a most general setting, which can be instantiated to almost any problem presentation in computer science (see [45, ch. 3] for an environment hierarchy). For instance, by using appropriate environment classes we can evaluate learning abilities, planning abilities, visual abilities, deductive abilities, etc. Note that a meaningful choice of each environment is much more effective than choosing them arbitrarily. This can replace or complement those evaluations based on repositories and it is certainly more effective than choosing all the environments or selecting them by a (universal) distribution, as in [46] (including some improvements over the distribution, such as the one introduced by [39]).

A second application is the assessment of ill-specified problems and interfaces, i.e., when we do not have a complete description of the environment or interactive task at hand. A complete specification of the problem would be necessary for the direct calculation of the (Kolmogorov) complexity of the environment, but it is not for the indicators, plots and measures of difficulty based on (actions or) policies used here. This means that we can apply these techniques to real-world scenarios, stochastic systems, chaotic systems, biological systems, etc. This includes the assessment of a wide variety of multi-agent systems, where the behaviour of the other agents is unknown, with some of them possibly being natural agents (animals or humans), including real environments, games, robotic environments, swarms, etc. This is of course complementary to (and helpful for) the increasing number of competitions in artificial intelligence [59,1], where the score range and weight of each problem could be derived from the difficulty measures and functions introduced in this paper.

A third application is to help develop better heuristics, especially in reinforcement learning. This can be done in two ways. First, we can analyse the kinds of environments (according to their response curves) that a given heuristic can solve. This can provide useful information about the exact problems where the heuristic is struggling and also whether there is a neat correlation with the indicators of difficulty. Second, we can customise the techniques used here by changing the probability w of each policy according to some of the features of the heuristic and see how the results change. In any of both ways, the relation of our approach with Levin's search must be fundamental here.

A fourth application is a better understanding of the notion of difficulty and its relation to existing problem solvers, especially in the area of natural and artificial evolution. Since our approach is based on the evaluation of a population of agents, we can specialise this population according to a given natural population (as psychometrics does) or to some theoretical population (in theoretical biology, artificial life or evolutionary computation). How the distributions and measures of difficulty evolve generation after generation (when facing the same or different problems) could provide valuable information about the evolutionary process itself.

The contributions and applications mentioned above also carry some limitations. One limitation is the use of Kolmogorov complexity, which has two important problems: it is incomputable (and needs to be approximated) and it involves constants that depend on the reference machine, which may be very important for small objects. Fortunately, there has been a significant improvement in the empirical approximation of Kolmogorov complexity [14,64], and new methods have been devised that are less sensitive to the reference machine. A second limitation is the agreement on a policy language with the capability of describing the representation in a powerful but unbiassed way. The use of an arbitrary set of policies from state-of-the-art systems (or even competitors) is not a good solution, as the population becomes poorly comprehensive and too subjective. So a policy language has to be agreed in order to generate distributions, measures and plots. A third limitation is the computational effort required to calculate the response curves. Although the plots and measures for the minimalistic setting used in this paper can be calculated in a matter of minutes with a personal computer, things may grow exponentially for more elaborated environment classes and policy languages. The development of tighter bounds that could be expressed in closed-form for the case without replacement is a promising direction to be explored in the future. Nonetheless, it is important to state that these evaluations do not have to be done in real time. They can be done beforehand and the results can be stored for successive applications of each environment. This is what psychometricians do; they construct a library or catalogue of items, whose response curves have been previously estimated (not effortlessly), and they can use them repeatedly on demand. Since our approach is not specialised to any kind of heuristic, environments that have already been evaluated can be used for a broader range of situations, also making the initial investment more beneficial. Once the distributions, difficulties and plots are derived, the evaluation of a new agent for an environment only requires one simulation. In fact, the estimation of its ability for an environment class will usually require a very small set of environments, if an adaptive testing protocol is used that takes advantage of the characteristics of each environment in terms of difficulty and discriminating power. In any case, the development of other ways of deriving environment response curves (under certain conditions or with simpler expressions) is encouraged in order to make their derivation easier and more efficient.

In addition, many new things can be explored, some of them related to the issues above. For instance, we could use better approximations of Kolmogorov complexity or some other variants (such as Levin's Kt, already used in [38,30]), leading to possibly more accurate views of difficulty, since time must be taken into account more explicitly from the beginning. The way in which the generation of policies is done could be rethought in order to resemble the way the coding theorem method works and get the estimation of Kolmogorov complexity (or any variant) directly, without further processing. We have already used random policies in our case studies. However, the introduction of a (pseudo-)random instruction in the instruction set of the policy language is (partially) at odds with the use of Kolmogorov complexity but would lead to interesting insights, especially in the case of matching pennies or other problems (e.g., predator and prey) where occasionally acting randomly may be beneficial.

Other settings have to be explored, such as the use of other kinds of agents, described with different (and universal) policy languages (e.g., [51,13]), using other kinds of environments, etc. For instance, we are considering applying this to mazes (in comparison with [62]), the matching pennies game (in relation to [37] and other normal-form games in game theory) or generalisations of cellular automata, such as Random Boolean Networks [26], in a setting that would highly resemble the one introduced in [33].

There is also an interesting work to be done in terms of indicators and graphical representations. For instance, we have used an empirical approach to the derivation of the environment response curve, but some other IRT models of difficulty and discriminating power ([22,23]) could be adapted to our case.

Overall, while there has been an enormous effort and significant progress in understanding what complexity is and how it emerges, the question about what difficulty is and how it emerges is more elusive. We hope that this work is a step forward towards a better understanding and measurement of the difficulty and discriminating power that an environment represents for a population of agents.

Acknowledgements

I thank the reviewers for their comments, especially those aiming at a clearer connection with the field of multi-agent systems and the suggestion of better approximations for the calculation of the response curves. The implementation of the elementary cellular automata used in the environments is based on the library 'CellularAutomaton' by John Hughes for R [57]. I am grateful to Fernando Soler-Toscano for letting me know about their work [65] on the complexity of 2D objects generated by elementary cellular automata. I would also like to thank David L. Dowe for his comments on a previous version of this paper.

This work was supported by the MEC/MINECO projects CONSOLIDER-INGENIO CSD2007-00022 and TIN 2010-21062-C02-02, GVA project PROMETEO/2008/051, the COST - European Cooperation in the field of Scientific and Technical Research IC0801 AT, and the REFRAME project, granted by the European Coordinated Research on Long-term Challenges in Information and Communication Sciences & Technologies ERA-Net (CHIST-ERA), and funded by the Ministerio de Economía y Competitividad in Spain (PCIN-2013-037).

Appendix. Proofs

Proof (proposition 1) If $N = 1$ we just draw one element, and with $\gamma = 1$ we have:

$$\begin{aligned} q_{\text{pos}}(1, \Omega, w, 1) &= \Pr\left[\max_{\pi \in \Omega||_{w,1}} \{R(\pi)\} \geq (1-1)(R_{\max} - R_{\text{mean}}) + R_{\text{mean}}\right] \\ &= \Pr\left[\max_{\pi \in \Omega||_{w,1}} \{R(\pi)\} \geq R_{\text{mean}}\right] \end{aligned}$$

If the mean and the median match then there are exactly $|\Omega|/2$ elements above R_{mean}, so the probability that taking just one is greater than or equal to R_{mean} is exactly $1/2$ because we have assumed independence.

Proof (proposition 2) The first one follows directly from proposition 1. For $\gamma = 0$, we have that q_{pos} simplifies to $\Pr(R_{\max} \in \Omega||_{w,N})$. Since w is independent of $R_{\max}$ by assumption, and there is only one policy for $R_{\max}$, we need to draw half of the elements of Ω in order to have exactly probability $1/2$.

Proof (proposition 3) The probability of π^*_{pos} appearing in a sample of size N is greater than or equal to $1-(1-w(\pi^*_{\text{pos}}))^N$ (this would be the probability with replacement). Then, since this policy is one of those reaching the maximum reward in Ω, we have that:

$$\begin{aligned} q_{\text{pos}}(0, \Omega, w, N) &= \Pr\left[\max_{\pi \in \Omega||_{w,N}} \{R(\pi)\} \geq (1-\gamma)(R_{\max} - R_{\text{mean}}) + R_{\text{mean}}\right] \\ &\geq 1 - (1 - w(\pi^*_{\text{pos}}))^N \end{aligned}$$

Since q_{pos} is non-decreasing with N, in order to calculate N we make q_{pos} equal to $1/2$:

$$\begin{aligned} q_{\text{pos}} = \quad 1/2 &\geq 1 - (1 - w(\pi^*_{\text{pos}}))^N \\ (1 - w(\pi^*_{\text{pos}}))^N &\geq 1/2 \\ (1 - \tfrac{2^{-k^*}}{w_{\text{all}}})^N &\geq 1/2 \end{aligned}$$

Now we use the following variable change $t = 1 - \frac{2^{-k^*}}{w_{\text{all}}}$. Since $k^* > 0$ we have that $0 < t < 1$. As a logarithm with base between 0 and 1 is a decreasing function, we have that the sign of the inequality changes if we apply this logarithm, so we have:

$$\begin{aligned} N &\leq \log_t(1/2) = \frac{\log_2(1/2)}{\log_2 t} \\ N &\leq \frac{-1}{\log_2 t} \end{aligned} \tag{15}$$

We see now that

$$\begin{aligned} \frac{-1}{\log_2(t)} &\leq \frac{1}{1-t} \\ \log_2(t) &\leq -1 + t \\ \frac{\ln(t)}{\ln(2)} &\leq -1 + t \\ \ln(t) &\leq (-1+t)\ln(2) \end{aligned}$$

is true because for $0 < t < 1$ we have that $\ln(t) \leq -1 + t$ and $(-1+t) < (-1+t)\ln(2)$ since $(-1+t)$ is negative. Using this in Eq. 15 and undoing the variable change[11] we have:

$$\begin{aligned} N &\leq \frac{-1}{\log_2(t)} \leq \frac{1}{1-t} \\ N &\leq \frac{1}{\frac{2^{-k^*}}{w_{\text{all}}}} = (2^{k^*})w_{\text{all}} = 2^{k^*} \sum_{\pi' \in \Omega} 2^{-K(\pi')} \end{aligned}$$

In fact, it can also be shown that this bound would be tight (if we used replacement in the sample), because for a slightly smaller k, i.e., $k^* - 1$, we would have $(1 - \frac{2^{-(k^*-1)}}{w_{\text{all}}})^N$. Figure 20 illustrates how these two functions would make tight bounds.

[11] In fact, until this point, this holds for every probability distribution w.

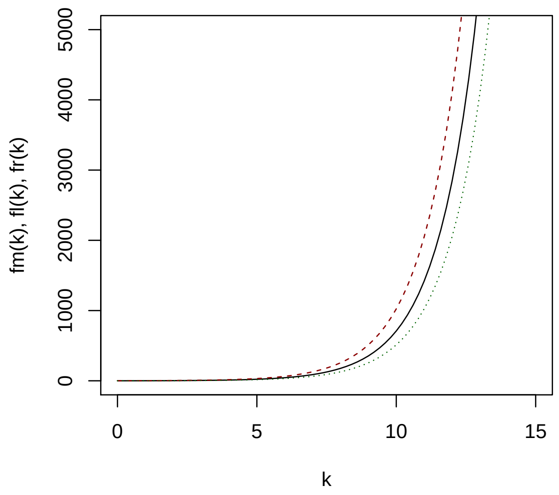


Fig. 20 Example of the approximation made by Eq. 15. If instead of $fm = \frac{-1}{\log_2(1-\frac{2^{-k}}{w_{\text{all}}})}$, we use $fr = (2^k)w_{\text{all}}$ and $fl = (2^{k-1})w_{\text{all}}$, fm lies tightly in between. The plot shows these three functions (fm solid, fr dashed, fl dotted), all with $w_{\text{all}} = 1$.

References

1. J. Anderson, J. Baltes, and C.T. Cheng. Robotics competitions as benchmarks for ai research. *The Knowledge Engineering Review*, 26(01):11–17, 2011.
2. D. Andre and S.J. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 119–125. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.
3. L. Antunes, L. Fortnow, D. van Melkebeek, and N. V. Vinodchandran. Computational depth: Concept and applications. *Theoretical Computer Science*, 354(3):391 – 404, 2006. Foundations of Computation Theory (FCT 2003), 14th Symposium on Fundamentals of Computation Theory 2003.
4. M. H. Ashcraft, R. D. Donley, M. A. Halas, and M. Vakali. Chapter 8 working memory, automaticity, and problem difficulty. In Jamie I.D. Campbell, editor, *The Nature and Origins of Mathematical Skills*, volume 91 of *Advances in Psychology*, pages 301 – 329. North-Holland, 1992.
5. N. Ay, M. Müller, and A. Szkola. Effective complexity and its relation to logical depth. *Information Theory, IEEE Transactions on*, 56(9):4593–4607, Sept.
6. D. M. Barch, T. S. Braver, L. E. Nystrom, S. D. Forman, D. C. Noll, and J. D. Cohen. Dissociating working memory from task difficulty in human prefrontal cortex. *Neuropsychologia*, 35(10):1373–1380, 1997.
7. R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason.* Wiley. com, 2007.
8. C. Boutilier, R. Reiter, M. Soutchanski, S. Thrun, et al. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the National Conference on Artificial Intelligence*, pages 355–362. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000.
9. L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008.
10. G. J. Chaitin. Algorithmic information theory. *IBM J. Res. Develop.*, 21:350–359, 1977.
11. F. B. Chedid. Sophistication and logical depth revisited. In *Computer Systems and Applications (AICCSA), 2010 IEEE/ACS International Conference on*, pages 1–4. IEEE, 2010.
12. P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-1991*, pages 331–337, 1991.
13. M. Dastani. 2APL: a practical agent programming language. *Autonomous agents and multi-agent systems*, 16(3):214–248, 2008.
14. J. P. Delahaye and H. Zenil. Numerical evaluation of algorithmic complexity for short strings: A glance into the innermost structure of randomness. *Applied Mathematics and Computation*, 2011.
15. D. L. Dowe. Foreword re C. S. Wallace. *Computer Journal*, 51(5):523 – 560, Sept 2008. Christopher Stewart WALLACE (1933-2004) memorial special issue.
16. D. L. Dowe and J. Hernández-Orallo. IQ tests are not for machines, yet. *Intelligence*, 40(2):77 – 81, 2012.
17. D. Z. Du and K. I. Ko. *Theory of computational complexity*, volume 58. Wiley-Interscience, 2011.
18. A. E. Elo. *The rating of chessplayers, past and present*, volume 3. Batsford London, 1978.
19. S. E. Embretson and S. P. Reise. *Item response theory for psychologists.* Lawrence Erlbaum, 2000.
20. N. Fatès and V. Chevrier. How important are updating schemes in multi-agent systems? an illustration on a multi-turmite model. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 533–540. International Foundation for Autonomous Agents and Multiagent Systems, 2010.
21. J. Ferber and J. P. Müller. Influences and reaction: a model of situated multiagent systems. In *Proceedings of Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 72–79, 1996.

22. P. J. Ferrando. Difficulty, discrimination, and information indices in the linear factor analysis model for continuous item responses. *Applied Psychological Measurement*, 33(1):9–24, 2009.
23. P. J. Ferrando. Assessing the discriminating power of item and test scores in the linear factor-analysis model. *Psicológica*, 33:111–139, 2012.
24. K. Arai G.A. Kaminka, I. Frank and K. Tanaka-Ishii. Performance competitions as research infrastructure: Large scale comparative studies of multi-agent teams. *Autonomous Agents and Multi-Agent Systems*, 7(1–2):121–144, 2003.
25. I.P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70(1):335–345, 1994.
26. C. Gershenson and N. Fernandez. Complexity and information: Measuring emergence, self-organization, and homeostasis at multiple scales. *Complexity*, 2012.
27. S. Gruner. Mobile agent systems and cellular automata. *Autonomous Agents and Multi-Agent Systems*, 20(2):198–233, 2010.
28. D. K. Hardman and S. J. Payne. Problem difficulty and response format in syllogistic reasoning. *The Quarterly Journal of Experimental Psychology*, 48(4):945–975, 1995.
29. J. He, C. Reeves, C. Witt, and X. Yao. A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability. *Evolutionary Computation*, 15(4):435–443, 2007.
30. J. Hernández-Orallo. Beyond the Turing Test. *J. Logic, Language & Information*, 9(4):447–466, 2000.
31. J. Hernández-Orallo. On the computational measurement of intelligence factors. In A. Meystel, editor, *Performance metrics for intelligent systems workshop*, pages 1–8. National Institute of Standards and Technology, Gaithersburg, MD, U.S.A., 2000.
32. J. Hernández-Orallo. Thesis: Computational measures of information gain and reinforcement in inference processes. *AI Communications*, 13(1):49–50, 2000.
33. J. Hernández-Orallo. A (hopefully) non-biased universal environment class for measuring intelligence of biological and artificial systems. In M. Hutter et al., editor, *Artificial General Intelligence, 3rd Intl Conf*, pages 182–183. Atlantis Press, Extended report at http://users.dsic.upv.es/proy/anynt/unbiased.pdf, 2010.
34. J. Hernández-Orallo and D. L. Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence*, 174(18):1508 – 1539, 2010.
35. J. Hernández-Orallo, D. L. Dowe, S. España-Cubillo, M. V. Hernández-Lloreda, and J. Insa-Cabrera. On more realistic environment distributions for defining, evaluating and developing intelligence. In J. Schmidhuber, K.R. Thórisson, and M. Looks (eds), editors, *Artificial General Intelligence 2011*, volume 6830, pages 82–91. LNAI series, Springer, 2011.
36. J. Hernández-Orallo, D. L. Dowe, and M. V. Hernández-Lloreda. Universal psychometrics: Measuring cognitive abilities in the machine kingdom. *Cognitive Systems Research*, 27:50–74, 2014.
37. J. Hernández-Orallo, J. Insa, D. L. Dowe, and B. Hibbard. Turing tests with Turing machines. In Andrei Voronkov, editor, *The Alan Turing Centenary Conference, Turing-100, Manchester, 2012*, volume 10 of *EPiC Series*, pages 140–156, 2012.
38. J. Hernández-Orallo and N. Minaya-Collado. A formal definition of intelligence based on an intensional variant of Kolmogorov complexity. In *Proc. Intl Symposium of Engineering of Intelligent Systems (EIS'98)*, pages 146–163. ICSC Press, 1998.
39. B. Hibbard. Bias and no free lunch in formal measures of intelligence. *Journal of Artificial General Intelligence*, 1(1):54–61, 2009.
40. H. H. Hoos. Sat-encodings, search space structure, and local search performance. In *1999 International Joint Conference on Artificial Intelligence*, volume 16, pages 296–303, 1999.
41. J. Insa-Cabrera, J. L. Benacloch-Ayuso, and Hernández-Orallo J. On measuring social intelligence: Experiments on competition and cooperation. In Joscha Bach, Ben Goertzel, and Matthew Iklé, editors, *AGI*, volume 7716 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 2012.
42. J. Insa-Cabrera, D. L. Dowe, S. España-Cubillo, M. V. Hernández-Lloreda, and J. Hernández-Orallo. Comparing humans and AI agents. In J. Schmidhuber, K.R. Thórisson, and M. Looks (eds), editors, *Artificial General Intelligence 2011*, volume 6830, pages 122–132. LNAI series, Springer, 2011.
43. D. E. Knuth. Sorting and searching, volume 3 of the art of computer programming, 1973.
44. K. Kotovsky and H. A. Simon. What makes some problems really hard: Explorations in the problem space of difficulty. *Cognitive Psychology*, 22(2):143–183, 1990.
45. S. Legg. *Machine Super Intelligence*. PhD thesis, Department of Informatics, University of Lugano, June 2008.
46. S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, 2007.
47. M. Leonetti and L. Iocchi. Improving the performance of complex agent plans through reinforcement learning. In *Proceedings of the 2010 International Conference on Autonomous Agents and Multiagent Systems: volume 1*, pages 723–730. International Foundation for Autonomous Agents and Multiagent Systems, 2010.
48. L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
49. L. A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15:285, 1986.
50. M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications (3rd ed.)*. Springer-Verlag, 2008.
51. C.K. Low, T.Y. Chen, and R. Rónnquist. Automated test case generation for bdi agents. *Autonomous Agents and Multi-Agent Systems*, 2(4):311–332, 1999.
52. M. G. Madden and T. Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21(3):375–398, 2004.
53. G. J. Mellenbergh. Generalized linear item response theory. *Psychological Bulletin*, 115(2):300, 1994.
54. F. Michel. Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents. *PhD thesis, Université des sciences et techniques du Languedoc, Montpellier*, 2004.
55. G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

56. P. Orponen, K. I. Ko, U. Schöning, and O. Watanabe. Instance complexity. *Journal of the ACM (JACM)*, 41(1):96–121, 1994.
57. R Team et al. R: A language and environment for statistical computing. *R Foundation for Statistical Computing, Vienna, Austria*, 2013.
58. H. A. Simon and K. Kotovsky. Human acquisition of concepts for sequential patterns. *Psychological Review*, 70(6):534, 1963.
59. S. Whiteson, B. Tanner, and A. White. The Reinforcement Learning Competitions. *The AI magazine*, 31(2):81–94, 2010.
60. M. Wiering and M. van Otterlo, editors. *Reinforcement Learning: State-of-the-art.* Springer, 2012.
61. S. Wolfram. *A new kind of science.* Wolfram media Champaign, IL, 2002.
62. Z. Zatuchna and A. Bagnall. Learning mazes with aliasing states: An LCS algorithm with associative perception. *Adaptive Behavior*, 17(1):28–57, 2009.
63. H. Zenil. Compression-based investigation of the dynamical properties of cellular automata and other systems. *Complex Systems*, 19(1):1–28, 2010.
64. H. Zenil. *Une approche expérimentale à la théorie algorithmique de la complexité.* PhD thesis, Dissertation in fulfilment of the degree of Doctor in Computer Science, Université de Lille, 2011.
65. H. Zenil, F. Soler-Toscano, J. P. Delahaye, and N. Gauvrit. Two-dimensional kolmogorov complexity and validation of the coding theorem method by compressibility. *arXiv preprint arXiv:1212.6745*, 2012.