Stochastic Tasks: Difficulty and Levin Search

José Hernández-Orallo

DSIC, Universitat Politècnica de València, Spain
jorallo@dsic.upv.es

Abstract. We establish a setting for asynchronous stochastic tasks that account for episodes, rewards and responses, and, most especially, the computational complexity of the algorithm behind an agent solving a task. This is used to determine the difficulty of a task as the (logarithm of the) number of computational steps required to acquire an acceptable policy for the task, which includes the exploration of policies and their verification. We also analyse instance difficulty, task compositions and decompositions.

Keywords: task difficulty, task breadth, Levin's search, universal psychometrics

1 Introduction

The evaluation of cognitive features of humans, non-human animals, computers, hybrids and collectives thereof relies on a proper notion of 'cognitive task' and associated concepts of task difficulty and task breadth (or alternative concepts such as composition and decomposition). The use of formalisms based on transition functions such as (PO)MDP (for discrete or continuous cases) is simple, but have some inconveniences. For instance, the notion of computational cost must be derived from the algorithm behind the transition function, which may have a very high variability of computational steps depending on the moment: at idle moments it may do just very few operations, whereas at other iterations it may require an exponential number of operations (or even not halt). The maximum, minimum or average for all time instants show problems (such as dependency on the time resolution). Also, the use of transition functions differs significantly in the way animals (including humans) and many agent languages in AI work, with algorithms that can use signals and have a control of time through threads (using, e.g., "sleep" instructions where computation stops momentarily).

The other important thing is the notion of response, score or return R for an episode. Apart from relaxing its functional dependency with the rewards during an episode, to account with a goal-oriented task, we consider the problem of commensurability of different tasks by using a level of tolerance, and deriving the notion of acceptable policy from it. While this seems a cosmetic change, it paves the way to the notion of difficulty —as difficulty does not make sense if we do not set a threshold or tolerance— and also to the analysis of task instances.

After these instrumental accommodations, the straightforward idea of difficulty as search effort is used. Difficulty is just the logarithm of the computational steps that are required to find an acceptable policy, including the execution of several possible policies and verifying them. This is in accordance with Levin's universal search [10,11], the notion of information gain [4] and the interpretation of the "minimal process for creating [something] from nothing" [12].

The notion of task instance difficulty is more controversial, as it usually assumes that it is relative to the task (e.g., '30+0' is an easy instance of the addition task) or even to the policy (e.g., 'sort gabcdef' is a very easy case for a particular sorting algorithm). Note that average-case complexity in complexity theory refers to how many computational steps are employed to solve a set of instances (with a distribution) given a particular algorithm —or for every possible conceivable algorithm. But one question that is not usually made is: How can we say that 'sort gabcdef' is easier than 'sort gdaefcb' without setting an algorithm or the distribution of algorithms?

The paper is organised as follows. Section 2 gives a setting for stochastic tasks, responses, difficulty and acceptability (using a tolerance level). Section 3 discusses whether the notion of task difficulty can be inherited for instances. Then we move to the notions of task composition and decomposition and their implications. Section 4 introduces a variant of Levin search that includes a new term into Kt, which is based on the number of repetitions that are needed to verify that a policy is ϵ-acceptable with some given confidence $1 - \delta$, à la PAC (Probabilistic Approximate Correct). Section 5 closes the paper.

2 Stochastic tasks, trials, responses and difficulty

Let us give the definition of asynchronous-time interactive systems. In an asynchronous-time interactive system, there is a common shared time (which can be discrete or continuous, and can be virtual or real). An *interactive system* is a machine with a program code, a finite internal discrete memory, one or more finite read-only discrete input tapes and one or more finite write-only discrete output tapes. The inputs of agents are called observations and the outputs are called actions. For tasks, it is the other way round. As special features, these machines have access to a read-only time measurement and a source of randomness (either by an additional random instruction or a random tape). The programs for tasks and agents are constructed with a Turing-complete set of instructions. The programs can be coded over a reference universal prefix Turing machine U. This makes this definition very close to probabilistic Turing machines. For the purpose of the analysis of computational steps, we consider an instruction or special state sleep(t), which sets the machine to sleep until time t.

Some tasks will also have intermediate rewards. Rewards are just given through another extra tape, and are interpreted as a natural number. Rewards are optional. In case they exist, the result of an episode may depend on the rewards or not. This is important, as the general use of rewards in reinforcement learning, especially with discounted reward or through averaging, gives the impression that the final result or response of an episode must always be an aggregation of rewards. For instance, in a maze, an agent may go directly to the exit and may require no reward. On the contrary, a more sluggish agent may require more positive indications and even with them cannot find the exit. Rewards can be just given to help in the finding of the solution. Finally, the agent is able to see the result or score of an episode at the end through another special tape. A final reward can be given instead of or jointly with the result.

The expected value of the response, return or result of agent π for task μ for a time limit τ is denoted by $\mathbb{R}^{[\tau]}(\pi, \mu)$. The value of τ will be usually omitted as it is understood that it is part of the description of the task μ. The $\mathbb{R}$ function always gives values between 0 and 1 and we assume it is always defined. If the agent goes into a non-halting loop and stops reacting, this is not perceivable externally and may even lead to some non-zero $\mathbb{R}$.

Now we need to extend the notation of $\mathbb{R}(\pi, \mu)$ to consider several instances of the same task. Each attempt of a subject on one of the task instances is a trial or episode. $\mathbb{R}^{[\mapsto\nu]}(\pi, \mu)$ returns the expected response of μ per trial with ν consecutive episodes or trials by the same agent π *without reinitialisation*. So actually it is not the same π each time, if the agent has memory. According to the task, the same instance can appear more than once, as in a sample with replacement. As the task can have memory, we can also have some tasks that are really working as if a no-replacement sampling were taking place. In order to do that, the task itself must keep track of the instances that have appeared or must use some kind of randomised enumeration. Also, tasks can be adaptive.

With $\mathbb{R}^{[\mapsto 1]}(\pi, \mu)$, or simply $\mathbb{R}(\pi, \mu)$, we denote that there is only one episode or trial. For instance, many tests are of this kind if items are completely unrelated, with no influence on the following ones, although it is more applicable when we consider that the agent has no memory (or is reinitialised between trials). In general, especially if the items are related, for every $\nu > 1$, we have that $\mathbb{R}^{[\mapsto\nu]}(\pi, \mu) \neq \mathbb{R}^{[\mapsto 1]}(\pi, \mu)$ unless the agent has no memory between episodes.

Our view of difficulty is "algorithmic", which is basically the computational steps required to build the policy algorithm, which depends on the tolerance level of the task, the interaction and hints given by the task, the algorithm length, its computation cost and its verification cost. The first thing we will require is the length of a policy or object x, denoted by $L(x)$. The second thing we will require is the computation steps taken by a policy. In synchronous environments, the sum or average of steps of all time cycles is not very meaningful. Another option is to calculate the maximum, as done in [7] with the so-called Kt^{max}. This is a very rough approximation, as one single peak can make this very large. Fortunately, here tasks are defined as asynchronous. When the agent needs to wait until a situation or time is met, if the instruction sleep(t) is used, these 'waiting' times are not considered for the computational steps. With this interpretation, the expected[1] execution steps of π per trial when performing task μ are denoted by $\mathbb{S}^{[\mapsto\nu]}(\pi, \mu)$ with a time limit (τ) given by the task for each trial. If at any moment π enters an infinite loop, then $\mathbb{S}^{[\mapsto\nu]}(\pi, \mu)$ is infinite. The third thing is about memory requirements (space). In this paper we will not consider space because (1) the use of n bits of memory requires at least n computational steps, so the latter are going to be considered anyway and (2) steps and bits are different

[1] This has to be 'expected' if we consider stochastic environments or agents.

units. The fourth thing is verification. When we discuss the effort about finding a policy, there must be some degree of certainty that the policy is reasonably good. As tasks and agents are stochastic, this verification is more cumbersome than in a non-stochastic case. We will discuss about this later on in the paper. For the moment, we will just combine the length of the policy and the computational steps, by defining $\mathbb{LS}^{[\mapsto\nu]}(\pi,\mu) \triangleq L(\pi) + \log \mathbb{S}^{[\mapsto\nu]}(\pi,\mu)$. Logarithms are always binary. We will explain later on why we apply a logarithm over $\mathbb{S}$. The fifth thing is the tolerance level of the task. In many cases, we cannot talk about difficulty if there is no threshold or limit for which we consider a policy acceptable. It is true that some tasks have a response function R that can only be 0 or 1, and difficulty is just defined in terms of this goal. But many other tasks are not binary (goal-oriented), and we need to establish a threshold for them. In our case, we can take 1 as the best response and set the threshold on $1-\epsilon$.

We now define acceptability in a straightforward way. The set of acceptable policies for task μ given a tolerance ϵ is given by

$$\mathcal{A}^{[\epsilon,\mapsto\nu]}(\mu) \triangleq \{\pi \,:\, \mathbb{R}^{[\mapsto\nu]}(\pi,\mu) \geq 1-\epsilon\} \tag{1}$$

Note that with a tolerance greater than 0 the agent can do terribly wrong in a few instances, provided it does well on many others.

And now we are ready to link difficulty to resources. This is usual in algorithmic information theory, but here we need to calculate the complexities of the policies (the agents) and not the problems (the tasks). A common solution, inspired by Levin's Kt (see, e.g., [10] or [11]), is to define:

$$Kt^{[\epsilon,\mapsto\nu]}(\mu) \triangleq \min_{\pi\in\mathcal{A}^{[\epsilon,\mapsto\nu]}(\mu)} \mathbb{LS}^{[\mapsto\nu]}(\pi,\mu) \tag{2}$$

Note that the above has two expectations: one in $\mathbb{LS}$ and another one inside $\mathcal{A}$. The interpretation of the above expression is a measure of effort, as used with the concept of computational information gain with Kt in [4].

An option as an upper-bound measure of difficulty would be $\hbar(\mu) \triangleq Kt^{[\epsilon,\mapsto\nu]}(\mu)$, for a finite ν and given ϵ. In general, if ν is very large, then the last evaluations will prevail and any initial effort to find the policies and start applying them will not have enough weight. On the contrary, if ν is small, then those policies that invest in analysing the environment will be penalised. It also requires a good assessment of the metasearch procedure to *verify* the policy so it can go to exploitation. In any case, the notion of difficulty depends, in some tasks, on ν. We will come back to the 'verification cost' later on.

3 Task instances, task composition and decomposition

Up to this point we have dealt with a first approach to *task* difficulty. A task includes (infinitely) many task instances. What about *instance* difficulty? Does it make sense? In case it does, instance difficulty would be very useful for adaptive tests, as we could start with simple instances and adapt their difficulty to the ability of the subject (as in adaptive testing in psychometrics).

The key issue is that instance difficulty must be defined *relative to a task*. At first sight, the difference in difficulty between 6/3 and 1252/626 is just a question of computational steps, as the latter usually requires more computational steps if a general division algorithm is used. But what about 13528/13528? It looks an easy instance. Using a general division algorithm, it may be the case that it takes more computational steps than 1522/626. If we see it easy is because there are some shortcuts in *our* algorithm to make divisions. Of course, we can think about algorithms with many shortcuts, but then the notion of difficulty depends on how many shortcuts it has. In the end, this would make instance difficulty depend on a given algorithm for the task (and not the task itself). This would boil down to the steps taken by the algorithm, as in computational complexity.

We can of course take a structuralist approach, by linking the difficulty of an instance to a series of characteristics of the instance, such as its size, the similarities of their ingredients, etc. This is one of the usual approaches in psychology and many other areas, including evolutionary computation, but does not lead to a general view of what instance difficulty really is. For the divisions above, one can argue that 13528/13528 is more regular than 1252/626, and that is why the first is easier than the second. However, this is false in general, as 13528^{13528} is by no means easier than any other exponentiation.

Another perspective is "the likelihood that a randomly chosen program will fail for any given input value" [2], like the population-based approach in psychology. For this, however, we would need a population[2]. The insight comes when we see that best policies may change with variable values of ϵ. This leads to the view of the relative difficulty of an instance with respect to a task *as the minimum* $\mathbb{LS}$ *for any possible tolerance of a policy such that the instance is accepted.* We denote by μ^σ an instance of μ with seed σ (on the random tape or generator). The set of all optimal policies for varying tolerances ϵ_0 is:

$$Opt_{\mathbb{LS}}^{[\mapsto\nu]}(\mu) \triangleq \left\{ \mathop{\arg\min}_{\pi\in\mathcal{A}^{[\epsilon_0,\mapsto\nu]}(\mu)} \mathbb{LS}^{[\mapsto\nu]}(\pi,\mu) \right\}_{\epsilon_0\in[0,1]} \tag{3}$$

And now we define the instance difficulty of μ^σ with respect to μ as:

$$\hbar^{[\epsilon,\mapsto\nu]}(\mu^\sigma|\mu) \triangleq \min_{\pi\in Opt_{\mathbb{LS}}^{[\mapsto\nu]}(\mu)\cap\mathcal{A}^{[\epsilon,\mapsto\nu]}(\mu^\sigma)} \mathbb{LS}^{[\mapsto\nu]}(\pi,\mu) \tag{4}$$

Note how the order of the minimisation is arranged in equations 3 and 4 such that for the many policies that only cover μ^σ but do not solve many of the other instances, these are not considered because they are not in $Opt_{\mathbb{LS}}$.

This notion of relative difficulty is basically a notion of consilience with the task. If we have an instance whose best policy is unrelated to the best policy for the rest, then this instance will not be covered until the tolerance becomes very low. Of course, this will depend on whether the algorithmic content of solving

[2] We could assume a universal distribution. This is related to the approach in this paper as the shortest policies have a great part of the mass of this distribution.

the instance can be accommodated into the general policy. This is closely related to concepts such as consilience, coherence and intensionality [3,5,4].

Now the question is to consider how we can put several tasks together. The aggregation of several responses that are not commensurate makes no sense This gives further justification to eq. 1, where $\mathcal{A}$ was introduced. Given two tolerance levels for each task we can see whether this leads to similar or different difficulties for each task. For instance, if the difficulties are very different, then the task will be dominated by the easy one. Given two stochastic tasks, the composition as the union of the tasks is meaningless, so we instead calculate a mixture. In particular, the composition of tasks μ_1 and μ_2 with weight $\alpha \in [0, 1]$, denoted by $\alpha\mu_1 \oplus (1-\alpha)\mu_2$, is defined by a stochastic choice, using a biased coin (e.g., using α), between the two tasks. Note that this choice is made for each trial. It is easy to see that if both μ_1 and μ_2 are asyncronous-time stochastic tasks, this mixture also is. Similar to composition we can talk about decomposition, which is just understood in a straightforward way. Basically, μ is decomposable into μ_1 and μ_2 if there is an α and two tasks μ_1 and μ_2 such that $\mu = \alpha\mu_1 \oplus (1-\alpha)\mu_2$.

Now, it is interesting to have a short look at what happens with difficulty when two tasks are put together. Given a difficulty function $\hbar$, we would like to see that if $\hbar(\alpha\mu_1 \oplus (1-\alpha)\mu_2) \approx \alpha\hbar(\mu_1) + (1-\alpha)\hbar(\mu_2)$ then both tasks are related, and there is a common policy that takes advantage of some similarities. However, in order to make sense of this expression, we need to consider some values of α and fix a tolerance. With high tolerance the above will always be true as $\hbar$ is close to zero independently of the task. With intermediate tolerances, if the difficulties are not even, the optimal policies for the composed task will invest more resources for the easiest 'subtask' and will neglect the most difficult 'subtask'. Finally, using low tolerances (or even 0) for the above expressions may have more meaning, as the policy must take into account both tasks.

In fact, there are some cases for which some relations can be established. Assume 0 tolerance, and imagine that for every $1 > \alpha > 0$ we have $\hbar(\alpha\mu_1 \oplus (1-\alpha)\mu_2) \approx \alpha\hbar(\mu_1)$. If this is the case, it means that we require the same effort to find a policy for both tasks than for one alone. We can see that task μ_1 *covers* task μ_2. In other words, the optimal policy for μ_1 works for μ_2. Note that this does not mean that every policy for μ_1 works for μ_2. Finally, if μ_1 covers μ_2 and vice versa, we can say that both tasks are equivalent.

We can also calculate a distance as $d(\mu_1, \mu_2) \triangleq 2\hbar(0.5\mu_1 \oplus 0.5\mu_2) - \hbar(\mu_1) - \hbar(\mu_2)$. Clearly, if $\mu_1 = \mu_2$ then we have 0 distance. For tolerance 0 we also have that if μ_2 has difficulty close to 0 but μ_1 has a high difficulty h_1, and both tasks are unrelated but can be distinguished without effort, then the distance is h_1.

Nonetheless, there are many questions we can analyse with this conceptualisation. For instance, how far can we decompose? There are some decompositions that will lead to tasks with very similar instances or even with just one instance. Let us consider the addition task μ_{add} with a soft geometrical distribution p on the numbers to be added. With tolerance 0, the optimal policy is given by a short and efficient policy to addition. We can decompose addition into μ_{add1} and μ_{add2}, where μ_{add1} contains all the summations $0 + x$, and μ_{add2}

incorporates all the rest. Given the distribution p, we can find the α such that $\mu_{add} = \alpha\mu_{add1} \oplus (1-\alpha)\mu_{add2}$. From this decomposition, we see that μ_{add2} will have the same difficulty, as the removal of summations $0+x$ does not simplify the problem. However, μ_{add1} is simple now. But, interestingly, μ_{add2} still covers μ_{add1}. We can figure out many decompositions, such as additions with and without carrying. Also, as the task gives more relevance to short additions because of the geometrical distribution, we may decompose the task in many one-instance tasks and a few general tasks. In the one-instance tasks we would put simple additions such as $1+5$ that we would just rote learn. In fact, it is quite likely that in order to improve the efficiency of the general policy for μ_{add} the policy includes some tricks to treat some particular cases or easy subsets.

The opposite direction is if we think about how far we can reach by composing tasks. Again, we can compose tasks *ad eternum* without reaching more general tasks necessarily. The big question is whether we can analyse abilities with the use of compositions and difficulties. In other words, are there some tasks such that the policies solving these tasks are frequently useful for many other tasks? That could be evaluated by looking what happens to a task μ_1 with a given difficulty h_1 if it is composed with any other task μ_2 of some task class. If the difficulty of the composed task remains constant (or increases very slightly), we can say that μ_1 covers μ_2. Are there tasks that cover many other tasks? This is actually what psychometrics and artificial intelligence are trying to unveil. For instance, in psychometrics, we can define a task μ_1 with some selection of arithmetic operations and see that those who perform well on these operations have a good arithmetic ability. In our perspective, we could extrapolate (theoretically and not experimentally) that this task μ_1 covers a range of arithmetic tasks.

4 Difficulty as Levin search with stochastic verification

In previous sections we considered the length of the policy and the logarithm of its computational time through their combination $\mathbb{LS}$, which finally led to the function $Kt^{[\epsilon, \mapsto\nu]}(\mu)$. As we argued, this is given by the realisation that in order to find a policy of length $L(\pi)$ we have to try approximately $2^{L(\pi)}$ algorithms if we enumerate programs from small to large (this is basically what Levin search does, see [11, pp. 577–580]). Considering that we can also gradually increase the computational steps that we devote for each of them, we get $2^{L(\pi)} \cdot \mathbb{S}(\pi, \mu)$, whose logarithm is represented by Kt. This is why we say that the unit of Kt is logarithm of computational steps.

If we try to extend this notion to tasks, the first, and perhaps most obvious and important difference with traditional Levin's universal search is that tasks are stochastic. Consequently, several trials may be needed for discarding a bad policy and the verification of a good one. Intuitively, a pair of problem and policy with low variability in the response (results) will be easier to be verified than another where results behave more stochastically.

Another difference is that we can think about a Levin search with memory (i.e., non-blind), as some of the observations on previous trials may be crucial. We need that the policies that are tried could also be search procedures over several trials. That means that Levin search actually becomes a metasearch, which considers all possible search procedures, ordered by size and resources, similar to other adaptations of Levin search for interactive scenarios [9,13].

As tasks are stochastic, we can never have complete certainty that a good policy has been found. An option is to consider a confidence level, such that the search invests as fewer computational steps as possible to have a degree of confidence $1 - \delta$ of having found an ϵ-acceptable policy. This clearly resembles a PAC (probably approximate correct) scenario [14].

The search must find a policy with a confidence level δ, i.e., $Pr(\pi \, \text{solves} \, \mu) \geq 1 - \delta$. If we denote the best possible average result (for an infinite number of runs) as r^*, we consider that a series of runs is a sufficient verification for a probably approximate correct (PAC) policy π for μ when:

$$Pr(r^* - \widehat{r} \leq \epsilon) \geq 1 - \delta \tag{5}$$

with $\widehat{r}$ being the average of the results of the trials (runs) so far.

First, we are going to assume that all runs take the same number of steps (a strong assumption, but let us remind that this is an upper limit), so the verification cost could be approximated by

$$\widehat{\mathbb{W}}^{[\epsilon,\delta]}(\pi,\mu) \triangleq \mathbb{S}(\pi,\mu) \cdot \mathbb{B}^{[\epsilon,\delta]}(\pi,\mu) \tag{6}$$

i.e., the expected number of steps times the expected number of verification bids.

The number of bids can be estimated if we have the mean and the standard deviation of the response for a series of runs. Assuming a normal distribution:

$$n \geq \frac{|z_{\delta/2}|^2 \sigma^2}{(\widehat{r} + \epsilon - r^*)^2} \tag{7}$$

In order to apply the above expression we need the variance σ^2. Many approaches to the estimation of a population mean with unknown σ^2 are based on a pilot or prior study (let us say we try 30 repetitions) and then derive n using the normal distribution and then use this for a Student's t distribution. Instead of this, we are going to take an iterative approach where we update the mean and standard deviation after each repetition. We consider the maximum standard deviation as a start (as a kind of Laplace correction) with two fabricated repetitions with responses 0 and 1.

Algorithm 1 is used in a modified Levin search:

Definition 1. *Levin's universal search for stochastic tasks and policies with tolerance ϵ, confidence level $1 - \delta$, and maximum response reference r^*. Given a task μ policies are enumerated in several phases, starting from phase 1. For phase i, we execute all possible policies π with $L(\pi) \leq i$ for $s_i = 2^{i-L(\pi)}$ steps each. We call function* VerifyNorm$(\pi, \mu, \epsilon, \delta, s_{max})$ *in Algorithm 1 with $s_{max} = s_i$. While an acceptable policy is not found we continue until we complete the phase and then to a next stage $i+1$. If an acceptable policy is found, some extra trials are performed before stopping the search for confirmation.*

Algorithm 1 VERIFICATION ALGORITHM (NORMALITY)

```
function VERIFYNORM($\pi$, $\mu$,$\epsilon$, $\delta$, $s_{max}$)    ▷ $s_{max}$ is the number of allowed steps
    $j \leftarrow 3$    ▷ We consider two first response with high variance
    $r \leftarrow 0 + 1$    ▷ One with value 0 and the other with value 1
    $s \leftarrow 0$
    $m_\pi \leftarrow \emptyset$    ▷ The algorithm $\pi$ can keep memory between trials. Initially empty.
    repeat
        $\langle r_j, s_j, m_\pi \rangle \leftarrow Run(\pi, m_\pi, \mu, s_{max} - s)$    ▷ One trial with remaining steps
        $s \leftarrow s + s_j$    ▷ Accumulate steps
        $r \leftarrow r + r_j$    ▷ Accumulate response
        $\widehat{r} \leftarrow \frac{r}{j}$    ▷ Average response
        $\widehat{\sigma}^2 \leftarrow \mathsf{Var}[r_1 \ldots r_j]$    ▷ Variance estimation
        $n_0 \leftarrow \frac{|z_{\delta/2}|^2 \widehat{\sigma}^2}{(\widehat{r} + \epsilon - r^*)^2}$
        if $j \geq n_0$ then
            if $\widehat{r} > r^* - \epsilon$ then return $\langle \mathsf{TRUE}, s \rangle$    ▷ Stop because it is verified
            else return $\langle \mathsf{FALSE}, s \rangle$    ▷ Stop because it is rejected
            end if
        end if
        $j \leftarrow j + 1$
    until $s \geq s_{max}$
    return $\langle \mathsf{FALSE}, s \rangle$
end function
```

Theorem 1. *For every μ and $\epsilon, \delta > 0$, if a maximum r^* exists achievable by a computable policy and it is given, then definition 1 conducts a finite search.*

Proof. As r^* is defined as the highest expected response for a resource-bounded policy, then there is a number of phases where the optimal policy is found and there are enough steps such that $\widehat{r}$ is becoming as closer to r^* so that $\widehat{r} + \epsilon - r^*$ approaches ϵ such that is verified $Pr(r^* - \widehat{r} \leq \epsilon) \geq 1 - \delta$. Note that as results are bounded and the highest variability is $\sigma^2 = 1/4$, so $n \sim \frac{|z_{\delta/2}|^2 \sigma^2}{(\epsilon)^2}$ is bounded.

In the end, what we want is to account for the variability of computational steps given by the variance of the response and its proximity to the threshold, as both things make verification more difficult. This is finally calculated as:

$$\mathbb{B}^{[\epsilon,\delta]}(\pi, \mu) \triangleq \frac{|z_{\delta/2}|^2 \mathsf{Var}[R(\pi, \mu)]}{(\mathbb{R}(\pi, \mu) + \epsilon - r^*)^2} \tag{8}$$

For both $\mathsf{Var}[R(\pi, \mu)]$ and $\mathbb{R}(\pi, \mu)$ we consider that we include two extra responses as a start, as done in Algorithm 1. And now the effort is rewritten as:

$$\log \mathbb{F}^{[\epsilon,\delta]}(\pi, \mu) \triangleq \log(2^{L(\pi)} \cdot \widehat{\mathbb{W}}^{[\epsilon,\delta]}(\pi, \mu)) = L(\pi) + \log \widehat{\mathbb{W}}^{[\epsilon,\delta]}(\pi, \mu) \tag{9}$$

For clarity, we can expand what $\mathbb{F}$ is by using eq. 6 and eq. 8:

$$\log \mathbb{F}^{[\epsilon,\delta]}(\pi, \mu) = L(\pi) + \log \mathbb{S}(\pi, \mu) \cdot \mathbb{B}^{[\epsilon,\delta]}(\pi, \mu) = L(\pi) + \log \mathbb{S}(\pi, \mu) + \log \mathbb{B}^{[\epsilon,\delta]}(\pi, \mu)$$

From here, we can finally define a measure of difficulty that accounts for all the issues that affect the search of the policy for a stochastic task:

$$\hbar^{[\epsilon,\delta]}(\mu) \triangleq min_{\pi} \log \mathbb{F}^{[\epsilon,\delta]}(\pi, \mu) \tag{10}$$

5 Conclusions

As we have mentioned during this paper, the notion of task is common in AI evaluation, in animal cognition and also in human evaluation. We set tasks and agents as asynchronous interactive systems, where difficulty is seen as computational steps of a Levin search, but this search has to be modified to cover stochastic behaviours. These ideas are an evolution and continuation of early notions of task and difficulty in [8] and [6] respectively. The relevance of verification in difficulty has usually been associated with deduction. However, some works have incorporated it as well in other inference problems, such as induction and optimisation, using Levin's Kt [4,12,1]. From the setting described in this paper, many other things could be explored, especially around the notions of composition and decomposition, task instance and agent response curves.

Acknowledgements: This work has been partially supported by the EU (FEDER) and the Spanish MINECO under grants TIN 2010-21062-C02-02, PCIN-2013-037 and TIN 2013-45732-C4-1-P, and by Generalitat Valenciana PROMETEOII2015/013.

References

1. Alpcan, T., Everitt, T., Hutter, M.: Can we measure the difficulty of an optimization problem? IEEE Information Theory Workshop (ITW) (2014)
2. Bentley, J.G.W., Bishop, P.G., van der Meulen, M.: An empirical exploration of the difficulty function. In: CSRS, pp. 60–71. Springer (2004)
3. Hernández-Orallo, J.: A computational definition of 'consilience'. Philosophica 61, 901–920 (2000)
4. Hernández-Orallo, J.: Computational measures of information gain and reinforcement in inference processes. AI Communications 13(1), 49–50 (2000)
5. Hernández-Orallo, J.: Constructive reinforcement learning. International Journal of Intelligent Systems 15(3), 241–264 (2000)
6. Hernández-Orallo, J.: On environment difficulty and discriminating power. Autonomous Agents and Multi-Agent Systems pp. 1–53 (2014), http://dx.doi.org/10.1007/s10458-014-9257-1
7. Hernández-Orallo, J., Dowe, D.L.: Measuring universal intelligence: Towards an anytime intelligence test. Artificial Intelligence 174(18), 1508 – 1539 (2010)
8. Hernández-Orallo, J., Dowe, D.L., Hernández-Lloreda, M.V.: Universal psychometrics: Measuring cognitive abilities in the machine kingdom. Cognitive Systems Research 27, 5074 (2014)
9. Hutter, M.: Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer (2005)
10. Levin, L.A.: Universal sequential search problems. Problems of Information Transmission 9(3), 265–266 (1973)
11. Li, M., Vitányi, P.: An introduction to Kolmogorov complexity and its applications (3rd ed.). Springer-Verlag (2008)

12. Mayfield, J.E.: Minimal history, a theory of plausible explanation. Complexity 12(4), 48–53 (2007)
13. Schmidhuber, J.: Gödel machines: Fully self-referential optimal universal self-improvers. In: Artificial general intelligence, pp. 199–226. Springer (2007)
14. Valiant, L.G.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)