Measuring Convergence Inertia: Online Learning in Self-Adaptive Systems with Context Shifts

Elvin Alberts and Ilias Gerostathopoulos

Vrije Universiteit Amsterdam, Amsterdam, the Netherlands
e.g.alberts@vu.nl
i.g.gerostathopoulos@vu.nl

Abstract. To deal with situations not specifically designed for (unknown-unknowns), self-adaptive systems need to learn the best – or at least good enough – action to perform in each context faced during operation. An established solution for doing so is through the use of online learning. The complexity of online learning however increases in the presence of context shifts – which are typical in self-adaptive systems. In this paper, we (i) propose a new metric, *convergence inertia*, to assess the robustness of reinforcement learning policies against context shifts, and (ii) use it to assess the robustness of different policies within the family of multi-armed bandits (MAB) to context shifts. Through an experiment with a self-adaptation exemplar of a web server, we demonstrate that inertia and the accompanying interpretation of the unknown-unknowns problem is a viable way to inform the selection of online learning policies for self-adaptive systems, since it brings the influence of context shifts to the forefront. In our experiment, we found that non-stationary MAB policies are better suited to handling context shifts in terms of inertia, although stationary policies tend to perform well in terms of overall convergence.

Keywords: online learning · self-adaptive systems · non-stationary · convergence inertia

1 Introduction

Self-adaptive systems (SAS) are able to react to changes in their environment and internal state to ensure a number of adaptation goals related to e.g. application performance, resource consumption, and failure avoidance [12]. While adaptation is mostly used to address known-unknowns, i.e. situations that one anticipates and designs a specific action/policy for, a highly challenging yet realistic class of SAS has to deal with unknown-unknowns, i.e. situations that are not entirely anticipated by the system designers [14]. Such situations can lead to a suboptimal system state where adaptation goals are no longer met. To deal with unknown-unknowns, SAS can apply online learning, i.e. learn the appropriate adaptation action at runtime out of a set of available actions.

Online learning in SAS typically takes the form of a reinforcement learning (RL) policy that continuously applies an action and monitors its reward (in terms

of overall utility of the system after the action is performed). For instance, an action may be to add or remove servers of a web application; the reward could be measured as the number of requests that can be timely served. After applying a number of actions, the RL policy gradually builds up the knowledge of which action to apply to maximize system utility.

A problem that arises when using RL for online learning in a SAS is that the SAS may undergo several context shifts during the learning phase. Following the web application example, the number of user requests may increase or decrease. Context shifts affect the reward distributions, which can interfere with the learning process as this is based on associating actions with reward values. Interference may have a positive effect if it reinforces prior knowledge and speeds up the convergence of the policy or a negative effect if it contradicts learned knowledge and slows down convergence. An example of the latter is a context shift that causes a previously optimal action to become suboptimal. In that case, the convergence to the new optimal action will be hampered because of the prior knowledge accumulated suggesting the previous optimal. We refer to this difference in the speed of convergence as *inertia*, as there is a resistance towards realizing the optimal relative to a clean start, specific to each policy.

In this paper, we focus on inertia and its effects on online learning in self-adaptive systems. In particular, we formulate the inertia metric and propose a way to measure it general to any (even non-RL) policy. To investigate its effects, we perform an experimental study using a number of RL policies belonging to the multi-armed bandits (MAB) family [13], which is a simplified category of RL algorithms. Our experiment uses SWIM [18], a SAS exemplar provided by the self-adaptive systems community that simulates an elastic web application. We compare several MAB policies that can deal with context shifts and the related inertia they incur in different ways: by just ignoring the change, by greedily exploring or assuming stationarity, by considering only a limited window of time in the past when evaluating actions, or by maintaining separate knowledge bases for a set of learned contexts through side information.

Our results show that stochastic policies that operate under the assumption of stationarity, such as `UCB Tuned` [2], can deceivingly converge well despite context shifts. However, when inertia is measured it is clear that non-stationary policies are better suited to handling context shifts. By quantifying the inertia an educated decision can be made when choosing RL policies for SAS systems which deal with non-stationary environments.

2 Background and Running Example

2.1 Online Learning in Self-adaptive Systems

Online learning has been proposed to remedy design uncertainty in self-adaptive systems (SAS) [9, 17]: instead of trying to enumerate all the possible situations – triggers for adaptation – and the corresponding actions, the idea is to let the system try out different actions in different situations and learn *at runtime* which action is the most appropriate in each situation.

With respect to the Monitor-Analyse-Plan-Execute over Knowledge (MAPE-K) loop typically used for structuring the self-adaptation logic [8] of a SAS, online learning can be employed within the Plan phase. In this setup, there exists a MAPE-K loop that monitors the managed system, analyzes the monitored data to determine whether a change is needed, plans the change and executes it (Fig. 1). This is done by using system models and/or rules shared via the common Knowledge base. If the planner has no available plans for a situation, it invokes online learning with a certain time budget (horizon) to generate such plan. Online learning then uses the Monitor and Execute phases of the outer loop and, instead of reacting to conditions by changing the system at runtime, it proactively performs actions to assess and rank them at runtime.

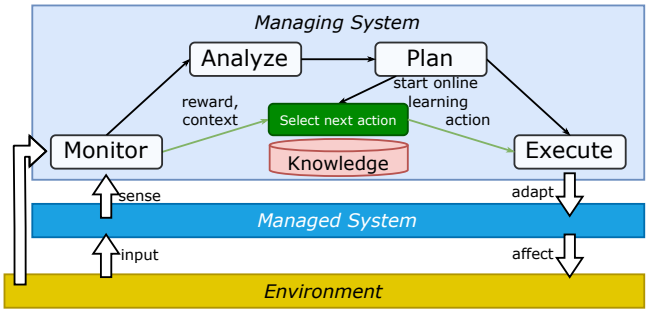


Fig. 1: Online learning invoked by the Plan phase of a self-adaptive system.

In this work, we focus specifically on the multi-armed bandits family of RL algorithms to perform online learning and compare their capability to learn optimal actions while the system undergoes context shifts. MAB policies are not considered 'full' RL [21], in that they only consider a single state. This entails that every action only affects its immediate reward (and not that of other actions), as there is no transition in states due to the action. Where the assumption of a single state can be placed on the system MAB policies provide sufficient solutions. These solutions are importantly also more accessible, and less complex and thus lightweight.

2.2 Overview of Multi-Armed Bandits

Multi-armed bandit (MAB) algorithms or *policies* (used henceforth) are a class of RL algorithms that deals with choosing between a set of k options called *arms* [13]. Formally, this setting corresponds to a Markov Decision Process with a single state and k actions. Compared to general RL, actions in MAB are assumed to not influence future states of the environment. As in general RL, an MAB policy balances *exploration* with *exploitation*: it tries to explore arms to gain knowledge about them while at the same time use the best-known arm

regularly enough to exploit the knowledge it has already gained. Each arm has an associated reward whose value at a time t is not known prior to selecting it. Arms are selected sequentially and their rewards are gradually revealed; an MAB policy prescribes which arm should be selected at each round. MAB policies try to minimize the *regret* they incur, i.e. the reward lost relative to that of the best arm at each round. Equivalently, they try to maximize the cumulative reward over all rounds. Formally, given k arms and sequences of rewards $X_{i,1}$,$X_{i,2}$,... associated with each arm i, the regret R_n of a policy after n plays I_1, ..., I_n is [4]

$$R_n = \max_{i=1,\ldots,k} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}$$

Different MAB policies address the exploration-exploitation tradeoff in different ways. For instance, *naive* policies such as ϵ-`greedy` and explore-then-commit rely on prior knowledge to control the amount of necessary exploration, while *stochastic* policies such as Upper Confidence Bound (UCB) and its variations assume that rewards follow certain statistical distributions.

As an example, Figure 2 depicts the main logic of `UCB Tuned`, which selects the arm with the maximum score calculated as the sum of the *estimated mean* (line 7) and a *confidence* value calculated based on the number of rounds the arm is played and the total number of rounds (line 10). To update its knowledge, `UCB Tuned` simply keeps a running sum per arm of the rewards received and the times chosen (lines 14-16). We can see that this policy prioritizes the selection of arms that have high rewards and have not used many times in the past.

```
1  class UCBTN(Bandit):
2    def get_next_arm(self):
3      return max(self.arms, key= lambda arm: self.get_score(arm))
4
5    def get_score(self, arm):
6      cum_rew, cum_sq_rew, n_k = self.knowledge[arm]
7      est_mean = cum_rew / n_k
8      est_variance = (cum_sq_rew / n_k) - square(est_mean)
9      V_k = est_variance + sqrt(2 * (log(self.round)/n_k))
10     confidence = sqrt((log(self.round)/n_k) * V_k)
11     return est_mean + confidence
12
13   def update_knowledge(self, reward):
14     self.knowledge[self.last_action][CUM_REWARD]+=reward
15     self.knowledge[self.last_action][CUM_SQ_REWARD]+=square(reward)
16     self.knowledge[self.last_action][N_K]+=1
```

Fig. 2: Python snippet of the `UCB Tuned` policy.

Certain MAB policies are specifically developed to deal with non-stationary environments and also consider the influence of context shifts on reward values. Representatives of the former are the Sliding Window and Discounted UCB policies [11], which perform UCB over a window of the last τ specified rounds or discount rewards by a factor γ as they age, respectively. A representative of *contextual MAB* is `EXP4` [3], a policy that uses a number of experts (instances of MAB policies), each able to deal with a certain context. When a context shift occurs, the policy eventually learns which expert to listen for that context to choose the next arm.

2.3 Running example: SWIM

SWIM [18] is a 'Simulator for Web Infrastructure and Management' representing at a high level the management of server infrastructure for a fictitious web location. The simulator allows real web traces to be replayed in simulated time, making it possible to adapt to hours of web traffic in a span of minutes. The behavior of SWIM is determined by two variables, its dimmer value and the number of servers deployed. The latter's maximum can also be configured by the user, while the dimmer is on a scale of [0,1] with the increments being user configured as well. The dimmer value determines the rate at which advertisements are included in the responses to requests, here we opt to always include ads in every response. In our experiment reported in Section 4 we use SWIM mostly as-is having added the ability to embed Python code to take advantage of simulated time rather than interacting with it in an external real-time fashion [1].

Strictly speaking, in SWIM the assumption of having an action influence only its immediate reward does not hold: some actions trigger a temporary secondary state where the servers have a backlog of requests to process. We have engineered as a part of our solution a means to detect this state and 'clean' the backlog to return to the original state the learners interact with. Where relevant we will refer to this change as the 'cleaning trick'.

3 Dealing with Context Switches in Online Learning

Unknown unknowns are a recognized challenge in self-adaptive systems research [14]. This entails the existence of environmental changes that are unanticipated due to a failure of imagination. These unanticipated events matter most when they interfere with achieving *explicit* (quantifiable in the degree to which they have been achieved) adaptation goals. For the purposes of MAB/RL policies such a goal can be interpreted as a 'reward' which evaluates choices made. Changes in the environment naturally influence the reward, even within anticipated scenarios e.g. circumstances dictate whether adaptation A or B is best. This closed feedback loop enables policies to learn which choices are suited to the environment. However, naive policies expect some stationarity, that over infinite time there is a singular optimal choice of which can be learned. To learn about these

choices there is some degree of exploration which is traded off with exploiting the best learned choice.

A problem arises when learned knowledge is abruptly invalidated. Learned choices will continue to be made by a policy based on false pretenses. This is different from a stochastic setting which may cause noise in the evaluation of choices. In the stochastic setting it is held that each choice has an unchanging true mean which in infinite time can be discovered by the policy. In a non-stationary environment these true means shift i.e. the reward-generating functions per arm change. The abrupt invalidation is what we refer to as a shift to a new context. This is one way in which the unknown-unknowns problem manifests itself in online learning. The response of a learner to this abrupt change is to (slowly) learn the new reward functions. The speed of this learning will depend on the accrued knowledge from the previous context and the policy-specific interaction with it. Combining these two factors is what we propose to measure and refer to as a policy's *inertia*.

We will now formalize the necessary concepts to determine the inertia. An RL learner or policy has a (potentially finite) horizon H of interactions (rounds) with which to interact with the environment. We consider the horizon to be divisible into a set R of time ranges indicated in square braces: $R = \{[s, e] | s, e \in \mathbb{Z} \wedge s < e \wedge e \leq H\}$, where s is the starting point of the time range and e the ending point. To evaluate the RL policy's performance within a time range $[s, e]$ a typical measure is convergence i.e. the frequency of choosing the optimal action – we denote this as $\text{conv}([s, e])$. Given that, we calculate inertia as the difference in convergence between two time ranges that have starting points $s1$ and $s2$ (with $s1 < s2$) and the same duration d:

$$\text{inertia}(s1, s2, d) = \text{conv}([s2, s2 + d]) - \text{conv}([s1, s1 + d])$$

The main idea is that if the two time ranges belong to the same context $c1 \subset R$ (blue context in Fig. 3), inertia provides a measure of how much the convergence of a policy in a context $c1$ is affected by other contexts occurring before it, compared to the "cold start" convergence obtained by setting $s1$ to 0 in the above formula.

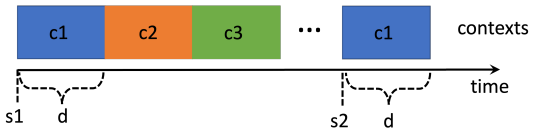


Fig. 3: Graphical representation of re-occuring context in inertia calculation.

For instance, if the convergence of a policy in the first occurrence of a context $c1$ is 60% and its convergence in the re-occurrence of $c1$ is 20%, the policy's inertia w.r.t. $c1$ is equal to -40%. Negative inertia values indicate a negative effect of

context shifts; the larger negative values, the more negatively a policy is affected by context shifts. Clearly, inertia values depend both on the policy used and on the particular setting – number and magnitude of context shifts.

In this paper, we focus on the way MAB policies deal with the negative effects of context shifts and measure and compare their inertia. For MABs, at each sequential round, one of the k arms available to the learner is chosen. The policy π maps previously chosen arms and their rewards to a new arm choice. Formally, it is a tuple $\pi = (H, I, X)$ where the *horizon* H is the number of total interactions with the environment, and the sequences I and X are the arms it learns to choose and the rewards received for each arm, respectively. The length or number of rounds of a time range is given by $\text{len}([s, e])) = e - s$. Given the above, convergence in MABs is measured as [21]:

$$\text{conv}([s, e]) = \frac{T_{i^*}([s, e])}{\text{len}([s, e])} \quad \text{where } i^* = \text{argmax } X_{i,t} \text{ for a given } [s, e] \in R.$$

with T_i referring to the number of times a particular arm i is chosen within a time range.

We will now closely examine and compare potential solutions to the problem of dealing with accrued inertia within the realm of MAB policies. In particular, we cover the following potential solutions to the problem:

Stationary/Naive Policies: Stationary and naive MAB policies operate under the assumption of a singular true mean to be learned per arm. Therefore, they do not anticipate context shifts as we describe them. The ability to realize a new optimal choice due to a shift is dictated by that exploration rate of the policy. Policies which are 'stochastic bandits' tend to have more higher exploration rates as they expect noise to obfuscate the means. When the context shifts, this is interpreted as noise, the previous context's reading were all noise deviating from the 'true' choice of the new context. This holds until the last context it faces. The crux is that the contexts need to last long enough to overcome the 'deviations' from the previous context to realize the true choice. If contexts change often then the policy is left chasing different true choices without ever converging to any of them. Policies also tend to reduce the exploration rate over time i.e. there is an assumption that the policy becomes more certain about each choice the more it has sampled it. This assumption does not hold for a context shift as the distribution being sampled from has essentially changed; it stands to reason then that as time progresses stationary policies become worse at handling context shifts. The stationary policies used in this paper are ϵ-`greedy` and `UCB Tuned` [2]. ϵ-`greedy` has a hyperparameter ϵ which dictates its rate of exploration. `UCB Tuned` uses the 'confidence' in the true means per arm to decide its exploration rate.

Non-stationary Policies: These policies are designed with context shifts in mind. They achieve this by operating with knowledge which has an expiry date. There is a continuous disregard for older knowledge with the aim of operating based

on the most recent findings. This is in the hope that when a context shift does happen, the readings from the new context start to outweigh that of the previous. If the new context is short-lived and there is a return to the previous then some knowledge of that context is still held. The prevalent issue is how quickly older knowledge should be discarded. This depends on some estimation of the frequency of changes to take place. When improperly tuned, one risks not being able to converge and exploit a context as adequately as a stationary policy or approximate a stationary policy by not recognizing context shifts in a timely manner. `Discounted UCB` [11] is used to represent a non-stationary policy for the experiment to follow. Its hyperparameter is a discount factor which lessens the weight of older knowledge in decision-making.

Contextual Policies: Contextual learners use 'side information' to recognize contexts. Stationary and strictly non-stationary policies, as covered in the previous two paragraphs, have the side effect of overwriting previous knowledge. This is due to the fact that they are *nonassociative*. Contextual policies instead associate side-information with learned knowledge to maintain distinct behavior per context. In essence, they learn a policy rather than an action. This eliminates the need to estimate prior the frequency of context shifts as with non-stationary policies. However, the stipulation of available side information is not a light one and restricts application to SAS systems. Depending on the policy this side information is expected in different forms. Two important contextual policies are `LinUCB` [15] and `EXP4` [3]. For `LinUCB` side information are 'features' which may characterize contexts. For this paper `EXP4` is used. Side information is used at design time to inform the selection and potential training of experts. In this paper side information is used to determine the number of expected contexts, with there then being one expert pre-trained per context. It is also an option to have these experts be untrained should no prior access to the expected contexts be available. The policy learns to associate each expert with its corresponding context at runtime.

4 Experiments

We have conducted an experiment aiming to answer the following research questions: **RQ1**: To what extent do different context lengths affect the convergence rate after a shift? Following from this, **RQ2**: How does convergence inertia compare as an indicator of a policy's ability to handle non-stationary environments?

In the experiment, two shifts take place between two distinct contexts A and B, with the pattern ABA. Each context has a different optimal arm choice with stationary but noisy reward distributions supporting each arm. We specifically focus on three scenarios which differ in the number of rounds each context in the sequence is active. Within each scenario, the policies are exposed to the same environment. Based on these policy-specific behavior these environments influence the knowledge accumulated which influences the inertia measured.

4.1 Setup

In our experiment, we use the SWIM exemplar and focus on arms that correspond to different numbers of servers: 3, 8, 13. These values are chosen to both cover the level of traffic that will be experienced in different contexts as well as provide a wider range of reward values. Without spreading out the reward values the suboptimality gaps between the options hinder the ability of the learner to distinguish options. To be able to better interpret the effect of each policy, all configurations have a dimmer value of 1.0 (i.e., always serving ads). The reward function used to evaluate each arm is based on the utility function originally included with SWIM [18] and is adapted to (i) always have the server cost manifested in the reward, (ii) introduce weights among the objectives, and (iii) introduce an upper response time threshold to have the minimum potential reward reflect the maximum, as described in our earlier work [1]. All three of these serve to increase the accuracy of reward values in evaluating the adaptations.

In terms of policies, we use the `Random` policy to serve as a baseline, `ϵ-greedy` as an extended baseline as it has fixed exploration rates as well as as a representative of stationary policies alongside `UCB Tuned` [2]. Further, `Discounted UCB` [11] represents a non-stationary policy and `EXP4` [3] a contextual policy (solving non-stationarity through association).

For the purposes of the experiment we vary the length of time that the contexts A and B are active within the pattern ABA. We use two lengths, $X = 6600$ sec and $Y = 1200$ sec which correspond to 110 and 20 rounds respectively as each round is after a 60 sec evaluation period. The two lengths represent a number of rounds in which each policy should converge towards the optimal are and should not respectively. Convergence is calculated as defined in Section 3. A policy has converged when the convergence is $\geq \frac{\lceil k/2 \rceil}{k}$ of sequential time ranges of fixed chosen length. In our case of 3 arms, it is then when the optimal arm has a convergence of ≥ 0.66. We refer to the first A in the sequence as first context, B as second context, and the second A as the third context.

Every scenario ends with a sequence of Y rounds, this is as we measure the effect of inertia which is relevant directly after the shift. We now elaborate on each chosen scenario:

- XXY: By having two longer contexts of X, the policies have sufficient time to converge to the optimal arms of the first two contexts. This should demonstrate the effect of an equal amount of rounds in influencing behavior learned in the first context.
- XYY: By having a longer context first context followed by a shorter experience of the second non-repeating context, we expect to see the learner be biased towards learning the third context (which is the same as the first). It is then interesting to see whether this holds for the policies we choose.
- YXY: By giving little time with the first context which returns after the second we expect an opposite effect of XYY. Policies may be able to still use the short first context experience to their advantage in the third context, but this may be at the cost of converging in the second context (which is $\geq 70\%$ of the scenario).

Context A and B both have uniformly distributed (noisy) traffic levels centered around 60 and 80 request/s respectively, fluctuating within a range of 10% both ways. We know empirically that the former can be handled with at least 3 servers, while the latter can be handled with 5 servers or more. Each policy is run 30 times per scenario, with the results shown being the average over those 30. This is to account for slight variation which is generated in the service capacity of the servers in SWIM, as well as those policies (like ϵ-`greedy`) which use a random factor in their decision logic. As covered in Section 3, inertia plays a role when context shifts happen. As defined, inertia is the difference in convergence over the same number of rounds between learning from scratch and learning after one or more context shifts. For example, if 20 rounds of context A from a clean start results in a convergence factor of 0.70 and after some arbitrary number of rounds of another context the same 20 rounds of context A have a convergence factor of 0.40 then the inertia is -0.30. Thus, negative values indicate a relatively poorer convergence than starting from scratch and vice versa.

4.2 Results

For Table 1 we are specifically interested in how the policies fare as a result of the shift in context. By experiencing these rounds inertia in realizing the optimal arm accumulates. The amount of inertia created depends directly on the policy as we control for environmental factors. We consider an ideal policy one which can finely balance the interests of converging to potentially long contexts (X) yet use knowledge from even short-lived contexts (Y) in case they reoccur.The policy would have low inertia yet high convergence.

The reward distributions of the previous context(s) influence the knowledge each policy builds. The actual knowledge is dictated by the arms the policy's behavior as it chooses arms at each round. Besides the arms chosen, the total number of elapsed rounds can also have an influence. It is clear from the results in both Tables 1 and 2 that the hyperparameters play a significant role. For each policy, the hyperparameter can be translated into greediness. Most straightforward is ϵ-`greedy`, which is more greedy the lower its ϵ value is as it explores less. `UCB Tuned` is a stationary policy meaning it assumes the optimal arm will not change, and therefore progressively more greedily converges towards the optimal arm with the rate depending on collected knowledge. As the γ value of `Discounted UCB` increases, it approximates `UCB Tuned`, thus for lower values of γ it is less greedy. For `EXP4` the learning rate η dictates how greedily new knowledge is used to decide which expert it should listen to.

The greediness across policies is a double-edged sword. As Table 2 indicates, `UCB Tuned` makes use of its greediness to achieve a high weighted average of convergence. However, as results in the appendix[1] clarify, this is mostly due to its high convergence in the first two contexts. The convergence rates suggest that the policy can easily eliminate the third choice (optimal in no contexts) and begin achieving $\frac{2}{3}$ convergence, as good as random between two choices. As

[1] https://github.com/EGAlberts/ISOLABandits/blob/main/ISOLA_appendix.pdf

Policy	Hyper-parameter	Convergence First 20 (Y)	Inertia XXY	Inertia XYY	Inertia YXY	AVG Inertia
`Random`	n/a	0.34	0.01	-0.04	0.00	-0.01
ϵ-`greedy` (stationary)	$\epsilon = 0.30$	0.60	-0.50	0.21	-0.40	-0.23
	$\epsilon = 0.40$	0.52	-0.38	0.20	-0.30	-0.16
	$\epsilon = 0.50$	0.51	-0.36	0.17	-0.29	-0.16
	$\epsilon = 0.60$	0.48	-0.27	0.09	-0.24	-0.14
	$\epsilon = 0.70$	0.47	-0.20	0.06	-0.22	-0.12
	$\epsilon = 0.80$	0.46	-0.22	0.00	-0.19	-0.14
`UCB Tuned` (stationary)	n/a	0.50	-0.50	-0.19	-0.25	*-0.31*
`Discounted UCB` (non-stationary)	$\gamma = 0.89$	0.40	0.00	0.01	0.00	0.00
	$\gamma = 0.92$	0.40	0.02	0.02	0.03	**0.02**
	$\gamma = 0.97$	0.45	0.04	0.03	0.07	**0.05**
	$\gamma = 0.99$	0.49	0.06	-0.18	0.06	-0.02
	$\gamma = 0.995$	0.50	-0.17	-0.25	-0.07	-0.16
	$\gamma = 0.997$	0.50	-0.47	-0.21	-0.15	*-0.28*
`EXP4` (contextual)	$\eta = 0.04$	0.41	-0.22	0.09	-0.29	-0.14
	$\eta = 0.10$	0.44	-0.36	0.16	-0.42	-0.21
	$\eta = 0.20$	0.40	-0.33	0.26	-0.38	-0.15
	$\eta = 0.40$	0.44	-0.37	0.19	-0.41	-0.20
	$\eta = 0.60$	0.46	-0.40	0.27	-0.45	-0.19
	$\eta = 0.80$	0.50	-0.45	0.17	-0.47	-0.25

Table 1: Inertia based on difference in convergence.

time progresses it attempts to converge towards the best of these two. A policy like ϵ-`greedy` does so immediately based on whichever comes out higher, this is also clear from the tables in the appendix. However, `UCB Tuned` is a stochastic policy and thus expects some noise which at times may suggest one choice being better than another despite their true means. It therefore needs to have enough confidence in a choice before it chooses to converge to it. The first shift from A to B aids in this by enlarging the gap between the two arms across scenarios. This leads to a minimum convergence of 0.80 in the second context, being highest (0.97) when the most time is afforded in XXY as can be seen in the appendix.

`Discounted UCB` anticipates that there may be shifts in the optimal arm. However it clearly does so at a cost, its robustness towards shifts directly influences its convergence towards an arm within the context. The `Discounted UCB` hyperparameter values which are not too close to `UCB Tuned` perform best when it comes to inertia, yet poorly when it comes to actually converging. This is a clearcut indicator of inertia corresponding to the ability to handle non-stationary environments. What is key to choosing between a non-stationary and stationary policy expectation a user has of the system and what is can afford. With a significant consistent frequency of shifts, averaging a convergence rate of ~ 0.50 becomes more attractive when compared to stationary policies. One needs to look no further than high effect of inertia on `UCB Tuned`, also reflected in its

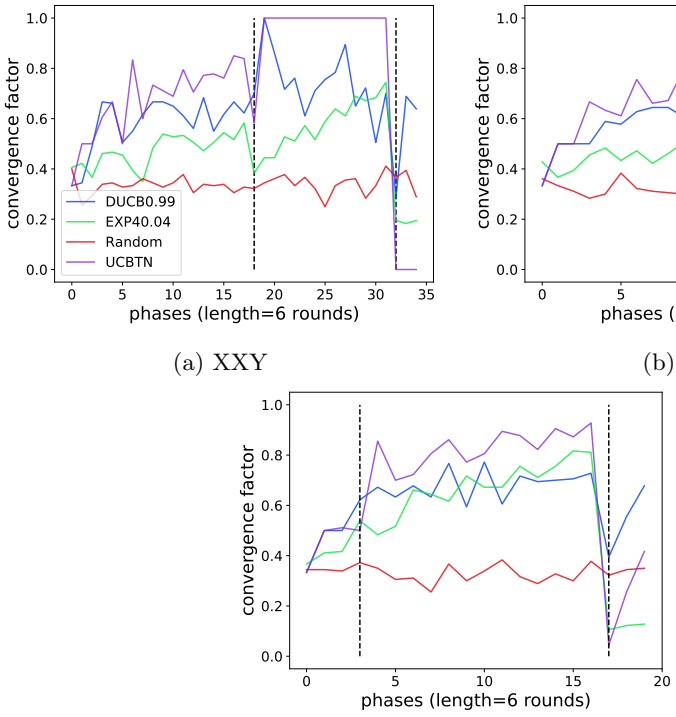
Policy	Hyperparameter	XXY	XYY	YXY	Average
`Random`	n/a	0.34	0.32	0.33	0.33
ϵ-`greedy` (stationary)	$\epsilon = 0.30$	0.59	0.70	0.44	0.58
	$\epsilon = 0.40$	0.56	0.67	0.42	0.55
	$\epsilon = 0.50$	0.54	0.60	0.39	0.51
	$\epsilon = 0.60$	0.48	0.54	0.37	0.46
	$\epsilon = 0.70$	0.44	0.50	0.36	0.43
	$\epsilon = 0.80$	0.41	0.43	0.35	*0.40*
`UCB Tuned` (stationary)	n/a	0.72	0.64	0.67	**0.68**
`Discounted UCB` (non-stationary)	$\gamma = 0.89$	0.40	0.39	0.42	*0.40*
	$\gamma = 0.92$	0.43	0.42	0.44	0.43
	$\gamma = 0.97$	0.50	0.50	0.53	0.51
	$\gamma = 0.99$	0.64	0.56	0.63	0.61
	$\gamma = 0.995$	0.67	0.60	0.65	0.64
	$\gamma = 0.997$	0.68	0.62	0.65	**0.65**
`EXP4` (contextual)	$\eta = 0.04$	0.48	0.47	0.54	0.50
	$\eta = 0.10$	0.51	0.53	0.59	0.54
	$\eta = 0.20$	0.54	0.60	0.61	0.58
	$\eta = 0.40$	0.56	0.61	0.62	0.60
	$\eta = 0.60$	0.57	0.64	0.62	0.61
	$\eta = 0.80$	0.57	0.65	0.65	0.62

Table 2: Weighted averages of convergence per scenario.

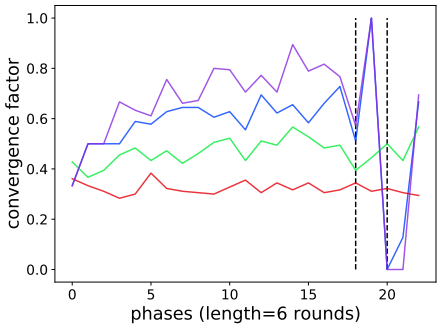
average convergence in the third context of only 0.18 (in the appendix) which is worse than even random selection. The challenge with using `Discounted UCB` is the necessity to tune it, the success of which depends on possibly unrealistic assumption one knows how many shifts will take place.

Contextual Bandits such as `EXP4` seek to remedy this by accounting directly for the existence of multiple contexts. In theory, if no switches take place, `EXP4` would at little cost behave as well as a stationary policy, while when facing changes pivoting to another expert with which to handle the new context. There remains a hyperparameter to tune however which is how eagerly it listens to a specific expert. Here we see the same trade-off as with other hyperparameters: switching experts too greedily (higher η) yields higher convergence, but also more inertia. This underlines that a more associative policy e.g. LinUCB may fare better as it removes the need for greed through classification. Yet, the assumption on side information accompanying such a policy would outgrow a fair comparison to the other policies we consider.

Fig. 4 shows the convergence factor of three selected policies over time, and the random policy as a baseline. The moments of context shifts are represented by the dashed lines, effectively dividing the plot into segments. The inertia per policy can be derived by focusing on the change in line trends after each shift. Figure 4a clearly shows the commitment of `UCB Tuned` to the optimal arm of the second context, and the consequence of this as the third context begins. In Figure



(a) XXY



(b) XYY

(c) YXY

Fig. 4: Convergence calculated over phases. The number of rounds in cases XYY and YXY slightly deviates due to the cleaning trick.

4b, the second context is short-lived seeing a reduced effect on the third context's convergence, and so too the positive reinforcement in `UCB Tuned`'s choice to converge allowing to better adapt to the third context. Lastly, Figure 4c shows `Discounted UCB`'s behavior clearly as it closely approximates the stationary `UCB Tuned` yet can handle the switch to the third context more gracefully. Throughout the plots it is clear that `EXP4` varies between being twice as indecisive as a stationary policy as in Figure 4b (due to its internal two stationary policies in the form of experts), and committing too heavily to one expert which is mimicking `UCB Tuned` as in Figure 4c.

Coming back to the research questions, we can answer **RQ1** as follows: It is clear that the longer a policy is exposed to a given context, the better it converges when that context reappears. For `UCB Tuned`, specifically, a greater factor than the length of individual contexts is the overall number of elapsed rounds. Over all policies, measuring performance through convergence rewards greed. Greedier policies converge heavily in their presumed contexts and outweigh their poor performance in other contexts when compared to non-stationary policies. This suggests at face-value that greedy, stationary, policies perform best de-

spite context shifts. **RQ2**'s answer shows that this is not actually the case when considering the big picture. Low inertia values are an indicator of average high convergence. Therefore the metric exposes undesirable behavior which otherwise is awarded by the traditional measure of convergence. When we take both inertia and convergence into account then a policy such as `Discounted UCB` with $\gamma = .99$ is shown to be a policy with more desirable long-term performance than e.g. `UCB Tuned` when dealing with frequent context shifts. A direct study of such a long-term scenario elucidating the consequences of poor inertia would concretize this in future work.

5 Discussion

Finding the right measure to compare policies: One of the struggles in our study was to determine a fair way to assess the performance of a policy under shifting contexts. Our initial idea was to use the degree of convergence to the optimal arm to compare policies and the way they progress in their learning. This serves well enough when considering stationary environments. However, once non-stationarity is introduced and realistic conditions are approximated more closely, the evolution of convergence is disrupted since the optimal arms may change. To measure the convergence taking into account such disruptions, one could refer e.g. to the average convergence after a context shift leading to a disruption. Alternatively, we chose to explore calculating the difference between the degree of convergence after a disruption relative to the convergence in that same environment with no prior knowledge (what we call inertia). This brings the influence of context shifts to the forefront in the metric. Where convergence is agnostic to context shifts besides it being disrupted by their presence, inertia is specific to their occurrence.

Performance of different policies in context shifts: Through our experiment, we confirm that the use of non-stationary policies is not a 'one size fits all' solution: depending on the magnitude (which we leave unexamined) and frequency of change, the policies need to be separately tuned. When exploring different hyperparameters in tuning the policies, inertia can serve as a useful metric for the success of policies specifically in handling context shifts. We did observe that the inertia of non-stationary policies is quite sensitive to the hyperparameter value chosen. Non-stationary policies such as `UCB Tuned` did not perform well in terms of accrued inertia (although they could reap the benefits of early convergence in longer contexts and overall have good performance when looking at the weighted average of their convergence).

How to select a policy: Our study shows that non-stationary environments do not necessarily call for non-stationary policies. In the end, it really depends on the effects of inertia (large or small) and on how much large inertia effects can be tolerated in a system. Some systems may be robust to policies which respond to context shifts by making continuous sequences of poor decisions as a stationary policy like `UCB Tuned` does. For these systems, stationary policies may be a more informed choice since they converge more greedily than non-stationary ones (as

also our results confirm). However, if a system cannot tolerate periods when the policy performs very poorly (suffering from large negative inertia effects) then non-stationary policies are a more reasonable choice since they better balance convergence speed with inertia effects.

Convergence Inertia Beyond MAB: Inertia is a metric that is not specific to MAB policies, but can be applied to any online planning strategy. Online planning requires knowledge which will inform policy behavior. This behavior is made time-sensitive due to the potential of context shifts. Inertia can be used to then also compare policies across paradigms and serve as a unifying metric to distinguish them. For example, a solution may recognize context shifts and choose to restart the learning process after every shift. This would result in an inertia of zero. This can be compared to an MAB policy which already has knowledge of the new context through seeing it before and therefore has a positive inertia, performing better than starting from nothing.

6 Related Work

According to a recent survey on the application of machine learning approaches for self-adaptive systems [7], reinforcement learning is a popular choice when it comes to online (or interactive) learning. Within reinforcement learning, most approaches have used temporal difference solutions such as Q-learning [9, 17], while recently value-based reinforcement learning has also been employed [16, 19]. In contrast, the application of MAB policies is less investigated.

Uses of MAB in SAS: In one of the first approaches that mapped a self-adaptation problem to MAB, Cabri et al. focused on endowing a multi-agent system with self-expression [5]. In particular, each collaboration pattern to expressed becomes an arm to be explored/exploited. In their work, they used three collaboration patterns, namely client/server, peer-to-peer, and swarm-based; measured reward in terms of observed application performance; and proposed and compared two custom strategies for maximizing the reward. While interesting, their approach neither considers out-of-the-box MAB policies with proven theoretical guarantees nor the specific problem of unknown-unknowns and associated convergence inertia.

Porter et al. employed MAB-based online learning to build distributed emergent software. [20]. In their approach, an arm is a composition configuration that specifies which components will run and where. Their online learning approach uses UCB1 to evaluate different configurations at runtime. Distinct environment classes are identified at runtime each with its own instance of UCB1. We believe that our work is complementary: Rather than externally imposing stationarity through environment classes, we evaluate policies that can inherently deal with environment switches, even if such switches are only indirectly considered (via their effect on rewards).

Dealing with Unknown-Unknowns in SAS: Kinneer et al. also tackle the issue of unknown-unknowns with applied to a system closely related to SWIM. Rather than using RL, the authors use genetic algorithms with 'plan reuse',

reusing previous knowledge for newer generations [10]. Compared to our work, their end-result is more applied, while the MAB policies we use are in principle generalizable to any architecture-based SAS.

Cardozo and Dusparic extend context-oriented programming to automatically codify contexts and associate strategies to handle them at runtime [6]. They do so by using RL options, a form of reinforcement learning which uses sequences rather than individual basic actions as the options to explore/exploit. These options are gathered by processing the execution trace at runtime. Key to their work is that system metrics are combined to explicitly define contexts, this is comparable to the solution by Porter et al. mentioned above. Our work differs in that we directly use RL to choose all actions, and do not require the overhead associated with COP to handle non-stationary environments.

7 Conclusion

In this paper, we focused on online reinforcement learning (RL) in self-adaptive systems as a technique to deal with unknown-unknowns – situations that the systems are not specifically designed for. We zoomed in on the problem of dealing with context shifts that interfere with the learning process by slowing down the convergence of RL policies. We proposed a new metric, convergence *inertia*, to capture such negative effect in the comparison of RL policies and performed an experimental study comparing RL policies belonging to the family of multi-armed bandits (MAB) in online learning of actions of a self-adaptive web server. We found that non-stationary policies are better suited to handling context shifts in terms of inertia, although stationary policies tend to perform well in terms of overall convergence. In the future, we would like to experiment with non-MAB RL policies (such as Q-learning) to better understand and assess the way they incur inertia. We would also like to use different self-adaptive systems to be able to better generalize our results.

References

1. Alberts, E.G.: Adapting with Regret: Using Multi-armed Bandits with Self-adaptive Systems. Master's thesis, University of Amsterdam (2022), https://scripties.uba.uva.nl/search?id=727497
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning **47**(2), 235–256 (May 2002). https://doi.org/10.1023/A:1013689704352
3. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: The Nonstochastic Multiarmed Bandit Problem. SIAM Journal on Computing **32**(1), 48–77 (Jan 2002). https://doi.org/10.1137/S0097539701398375
4. Bubeck, S.: Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. Foundations and Trends® in Machine Learning **5**(1), 1–122 (2012). https://doi.org/10.1561/2200000024

5. Cabri, G., Capodieci, N.: Applying Multi-armed Bandit Strategies to Change of Collaboration Patterns at Runtime. In: 2013 1st International Conference on Artificial Intelligence, Modelling and Simulation. pp. 151–156. IEEE, Kota Kinabalu, Malaysia (Dec 2013). https://doi.org/10.1109/AIMS.2013.31
6. Cardozo, N., Dusparic, I.: Auto-cop: Adaptation generation in context-oriented programming using reinforcement learning options. CoRR **abs/2103.06757** (2021)
7. Gheibi, O., Weyns, D., Quin, F.: Applying Machine Learning in Self-adaptive Systems: A Systematic Literature Review. ACM Transactions on Autonomous and Adaptive Systems **15**(3), 1–37 (Sep 2021). https://doi.org/10.1145/3469440
8. Kephart, J., Chess, D.: The Vision of Autonomic Computing. Computer **36**(1), 41–50 (2003)
9. Kim, D., Park, S.: Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 76–85 (May 2009). https://doi.org/10.1109/SEAMS.2009.5069076, iSSN: 2157-2321
10. Kinneer, C., Coker, Z., Wang, J., Garlan, D., Goues, C.L.: Managing uncertainty in self-adaptive systems with plan reuse and stochastic search. In: Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems. pp. 40–50. ACM, Gothenburg Sweden (May 2018). https://doi.org/10.1145/3194133.3194145
11. Kivinen, J., Szepesvári, C., Ukkonen, E., Zeugmann, T. (eds.): Algorithmic Learning Theory: 22nd International Conference, ALT 2011, Espoo, Finland, October 5-7, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6925. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24412-4
12. Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., Becker, C.: A survey on engineering approaches for self-adaptive systems. Pervasive and Mobile Computing **17**, 184–206 (Feb 2015). https://doi.org/10.1016/j.pmcj.2014.09.009
13. Lattimore, T., Szepesvári, C.: Bandit Algorithms. Cambridge University Press, 1 edn. (Jul 2020). https://doi.org/10.1017/9781108571401
14. de Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Göschka, K.M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezzè, M., Prehofer, C., Schäfer, W., Schlichting, R., Smith, D.B., Sousa, J.P., Tahvildari, L., Wong, K., Wuttke, J., Giese, H., Müller, H.A., Shaw, M.: Software Engineering for Self-Adaptive Systems: A Second Research Roadmap, pp. 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_1
15. Li, L., Chu, W., Langford, J., Schapire, R.E.: A Contextual-Bandit Approach to Personalized News Article Recommendation. Proceedings of the 19th international conference on World wide web - WWW '10 p. 661 (2010). https://doi.org/10.1145/1772690.1772758, arXiv: 1003.0146
16. Metzger, A., Kley, T., Palm, A.: Triggering Proactive Business Process Adaptations via Online Reinforcement Learning. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) Business Process Management, vol. 12168, pp. 273–290. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_16, series Title: Lecture Notes in Computer Science

17. Metzger, A., Quinton, C., Mann, Z.A., Baresi, L., Pohl, K.: Feature-Model-Guided Online Learning for Self-Adaptive Systems. arXiv:1907.09158 [cs] **12571**, 269–286 (2020). https://doi.org/10.1007/978-3-030-65310-1_20, arXiv: 1907.09158
18. Moreno, G.A., Schmerl, B., Garlan, D.: SWIM: an exemplar for evaluation and comparison of self-adaptation approaches for web applications. In: Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems. pp. 137–143. ACM, Gothenburg Sweden (May 2018). https://doi.org/10.1145/3194133.3194163
19. Palm, A., Metzger, A., Pohl, K.: Online Reinforcement Learning for Self-adaptive Information Systems. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) Advanced Information Systems Engineering, vol. 12127, pp. 169–184. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_11, series Title: Lecture Notes in Computer Science
20. Porter, B., Rodrigues Filho, R.: Distributed Emergent Software: Assembling, Perceiving and Learning Systems at Scale. In: 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO). pp. 127–136 (Jun 2019). https://doi.org/10.1109/SASO.2019.00024, iSSN: 1949-3681
21. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)