Comparing bandit algorithms in static and changing environments

An experimental study on the regret performance of bandit algorithms in various environments

Cody Michel Boon[1]

Supervisor: Julia Olkhovskaya[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Cody Michel Boon
Final project course: CSE3000 Research Project
Thesis committee: Julia Olkhovskaya, Ranga Rao Venkatesha Prasad

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

The aim of this paper is to show experimental data on the regret-based performance of various solver algorithms within a class of decision problems called Multi-Armed Bandits. This can help to more efficiently choose the algorithm most suited for an application and to reduce the amount of modification work needed to adapt the algorithm to fulfill its purpose. This is done because current research is largely based on theoretical analysis, strictly within a specific environment, using random data. These studies can be difficult to use to predict the performance of a practical implementation in a less strictly defined environment, but the amount of data on more directly comparable practical implementations is limited. The experiments involve several of the environments which some of the algorithms have been designed for, including a basic stochastic environment, a static linear contextual environment and two different changing contextual environments. Based on the experiments ran in these environments, we conclude that algorithms designed for stochastic environments tend to perform a lot worse in contextual environments than algorithms designed for linear contextual environments, and that all of the algorithms we tested that were designed for contextual environments tend to perform similarly in most cases. We also find that in most cases the contextual algorithms' performance relative to each other in changing environments is generally similar to their relative performance in static ones, even though the absolute regret behaviour tend to be different in the different environments. Our findings also include various more specific strengths and weaknesses for the individual algorithms.

1 Introduction

Multi-armed Bandit (MAB) problems are problems where one has to repeatedly pick between performing one out of two or more different actions, also called arms, with unknown (random) reward distributions with the objective to minimize (optimal policy) regret. Regret in this case defined as "the difference between the total expected reward using policy π for n rounds and the total expected reward collected by the learner over n rounds [1, p. 10]." In other words, a solver for an MAB problem has to repeatedly attempt make the best choice out of some options, but they do not know how the rewards of each option are determined.
Problems that can be represented as MAB problems are very common in relevant modern-day computing, occurring in areas like web development (e.g. ad placement and recommendation systems), machine learning and many more. While these problems are common, the exact environment in which they occur often differs and a specialized implementation is often required to solve them efficiently. This means there is high value in thoroughly understanding the effect of the problem environment on various algorithms designed to solve MAB problems, how well the algorithms adapt to variations in these environments in practice and what changes can generally be made to the algorithms to respond to these variations.
The primary source on the main concepts of MAB problems for this paper is the book "Bandit Algorithms" by Tor Lattimore and Csaba Szepesvári [1] which presents a lot of background information on these problems, some relevant environments and various common bandit algorithms.

The aim of this paper is to find the difference in regret based performance between various algorithms (see Table 1), depending on whether they are in an environment with static, perturbed or slowly changing reward functions.

Table 1: Algorithms considered in this research project

Algorithm	Intended purpose
UCB [1], [2], [3]	Basic MAB
EXP3 [4], [3]	Adversarial MAB
linUCB [5]	Linear contextual MAB
CW-OFUL [6]	Adversarial contextual MAB
SW-UCB [7]	Slowly changing reward MAB

The algorithms that will be considered are sourced from various papers in the field. The simplest, UCB, was first described by Tze-Leung Lai [2] and has been used as a basis for many other algorithms for MAB problems. A problem with UCB is that it does not hold up against an adversarial environment, as it is very predictable. This is what EXP3 [4] was designed for. While adversarial environments themselves are outside the scope of this paper, it is still valuable to know how EXP3 performs in other environments. At roughly the same time as the development of EXP3, Auer [8] and Abe et al. [9] developed and introduced the linear contextual bandit problem, where the arms share a reward vector but use their own context, which is visible to the solver algorithm, to determine the reward. Various algorithms have been developed to solve this version of the problem, but our focus will be on linUCB, introduced by Li et al. [10] and analysed by Chu et al. [5], the latter of which was used as a guide to implement the algorithm for this paper. Based on the last two environments, the linear contextual environment with adversarial corruptions was later created. While adversarial environments are outside the scope of this paper, they did inspire the creation of various algorithms. One of which is CW-OFUL [6], which uses weighted ridge regression, weighted with the inverse of the exploration bonus, to predict a corrupted θ^*. Lastly, we consider the work of Cheung et al. [7] on non-stationary environments and their algorithm SW-UCB, which is based on linUCB and uses a change budget β and a sliding window to predict a θ^* that changes over time. All of these papers have a strong focus on developing a (near) optimal algorithm for their respective intended environments and all except for

Li et al. [10] focus mainly on theoretical analysis.
What these papers lack is a broader, practical overview on the performance of their algorithms, especially in environments that do not perfectly represent what they were designed for, and a broader range of other algorithms to compare their performance to. While there is not enough time to make an extensive comparison between many of the known algorithms that can work in our selected environments in this paper, the chosen algorithms are chosen from various related larger categories and this should give some insight into how various similar algorithms will perform in an environment like this one.
In this paper we will first explain our testing environment and methodology in section 2, then in section 3 some noteworthy results will be shown and explained. Following that will be a brief explanation of our considerations relating to responsible research in section 4, after which some discussion points and potential open future research is described in section 5 and our conclusions in section 6.

2 Methodology

The testing environment for this study [11] was set up using Python 3.9, with the library SMPyBandits [3] as a basis. This has been expanded to support contextual bandits, contexts, contextual arms with static, changing or perturbed reward functions and all of the algorithms necessary for this research, as shown in Table 1.
All tests produce a ranking of each algorithm in text based on the final value of their average cumulative regret, and an "average cumulative regret over time" plot with 95% confidence bounds using MatPlotLib. Additionally, for most tests, plots of the rewards and semi-logarithmic plots of the regret have been stored (without confidence bounds). These additional plots will also be published but due to time constraints these will not be considered in this paper.
For every test, weight vector(s) $\vec{\theta}_t^*$ and context vectors $\vec{x}_{a,t}$ are normalized using their l_2 norm if their l_2 norm is greater than 1.0.
For every algorithm that uses parameters (all except for UCB), several different instances are used in each test environment, each with different parameters. All of these instances are included in the rankings of each test, but the plots only show the best of each algorithm for better readability.
The full list of algorithms with their parameters is the same for every test and can be found in Tables 2, 3, 4 and 5 in Appendix A. UCB is not listed there as its implementation in the library [3] has no parameters.
All tests are first ran with 10 repetitions and a time horizon of 3000. Some of the tests with noteworthy results will then be rerun with a higher number of repetitions and/or a higher horizon to produce clearer graphs and tighter bounds and to confirm trends. This will be visible in the graphs when it occurs: The number of repetitions is shown in the title of the graph and the time horizon is displayed on the x-axis.
After every test its setup, all result rankings and plots, including those from additional reruns, are stored. The full set has been published on Google Drive [12]. Please note that in the configuration documents, only the main diagonals of the contexts' covariance matrices are listed to save space. The rest of the values of these matrices are all zero. Furthermore, this set contains some graphs where the cumulative regret of all of the displayed algorithms is so low that the graphs show a lot of noise. This could be improved by various means, but that was not feasible in this study due to time constraints.

2.1 Process

First, tests are ran on UCB and EXP3 in a stochastic environment, using Gaussian reward distributions. As the other algorithms require contexts to run, which this environment does not have, they are excluded from this test.
The second stage of tests is ran with i.i.d. multivariate Gaussian distributed vectors as context vectors $\vec{x}_{a,t}$, comparing performance between various settings for $\vec{\theta}^*$, context means vectors $\vec{\mu}_a$, context covariance matrix Σ_a and all the algorithms' parameters.
After that, in stage three, the tests of stage two will be repeated, but the weight vector will occasionally be perturbed over the course of the tests with different alternative vectors and perturbation frequencies.
Stage four will be mostly the same as stage two and three, but the weight vector will be slowly changing instead, again with various amounts for different tests. This is done using a time based weighted average between two different weight vectors, which means the change in this environment is linear.

To run these tests, a large number of environments has been created. The full set of tests includes 9 stochastic environments, 45 static contextual environments, 756 perturbed contextual environments and 27 slow-changing contextual environments.
Due to time constraints, not all perturbed contextual environments have been run. Environments have been sampled from the set using an interval, taking every 10th environment and running it. The configuration data and every environment's index in the full set have been preserved to ensure repeatability and the option of running only the remaining tests later if desired.
After the tests are done, we will be comparing the performance of the various algorithms to each other. Statistics will be gathered on how often each one outperforms the others and how often they are outperformed in each of the environments, taking into account the 95% confidence bounds of their cumulative mean regret. If they can be clearly isolated, we will also show similarities and differences in the setups of environments where an algorithm distinguishes itself from the others in terms of performance.

3 Experiment configurations and Results

In the paragraphs below some noteworthy results and patterns are shown and explained for each of the tested environments. This includes both situations where algorithms perform notably different relative to each other and situations where

they perform very similarly. The full set of rankings, including the algorithm instances that were left out in the graphs, and the set of all graphs not shown in this paper, is published through Google Drive, as mentioned in the methodology section [12]. For the convenience of the reader, an enlarged version of the graphs in this section can also be found in Appendix B.

3.1 Stochastic environments

In stochastic environments, in experiments where all arms have similar or equal means and variances, UCB and EXP3 both perform very similarly, as can be seen in Figure 1. This is as expected because the algorithms don't have any valid metric they can use to pick a good arm, and therefore perform similar to an arbitrary random policy.

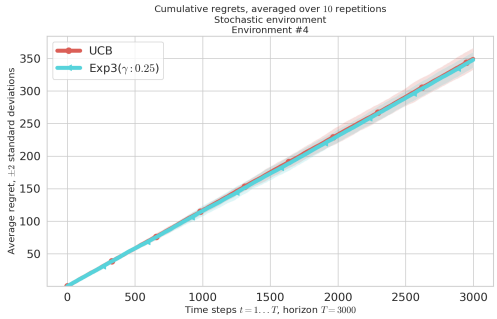

Figure 1: Stochastic environment with identical reward mean and variance for every arm

In experiments with different mean rewards for the arms and low reward variance UCB appears to perform significantly better than EXP3, like for example in Figure 2, where UCB and EXP3 seem to start converging at around the same time (roughly between t=100 and t=300). UCB has a sharper overall reduction in regret over time on average, which seems to be somewhat of a pattern in our stochastic environments.

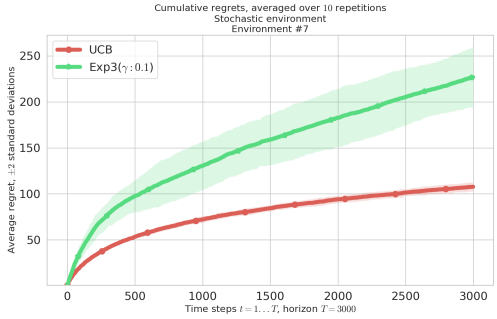

Figure 2: Stochastic environment with different reward mean for every arm and low reward variance

If the means are different but the variance is higher, UCB and EXP3 perform far more similar to each other within our given time horizon. This can be seen in Figure 3. The difference in this graph does not exceed the 95% confidence bound, so more computation may be needed to get a significant result in this situation.

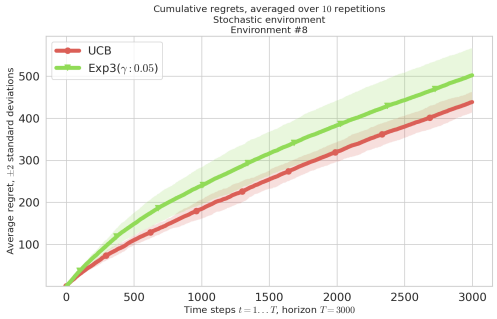

Figure 3: Stochastic environment with different reward mean for every arm and higher reward variance

3.2 Contextual environments

In contextual environments, and all other environments from this point onwards, all five algorithms are taken into account. For the behaviour of stochastic algorithms compared to contextual algorithms, four general patterns of behaviour have been found. These patterns seem mostly independent of θ^*.

The first of these patterns occurs in environments where the context means are all low, and environments where the means are all high and the context variances are uniformly high. In these environments, UCB and EXP3 behave very similar to an arbitrary random policy and the contextual algorithms perform very well in comparison, to the point of their regret not being clearly visible in a graph where stochastic algorithms are also shown. An example of this pattern is the environment in Figure 4.

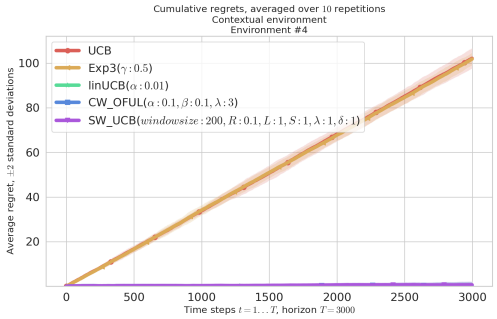

Figure 4: The first pattern in a contextual environment with identical low weights, identical low context means and low identical context variance

The second pattern occurs in our environments where the context means are high and the context variances are very different for each arm, and in all environments with very

different context means for each arm and any configuration of context variances. In these situations, the stochastic algorithms tend to have a lot of variance in their cumulative regret between repetitions, but their regret over time does converge to a lower trend within the time horizon of 3000 steps, generally within the first 1000 steps. Their behaviour compared to each other is still similar in these environments. The contextual algorithms perform similarly to the previous example. An example of this pattern can be seen in Figure 5.

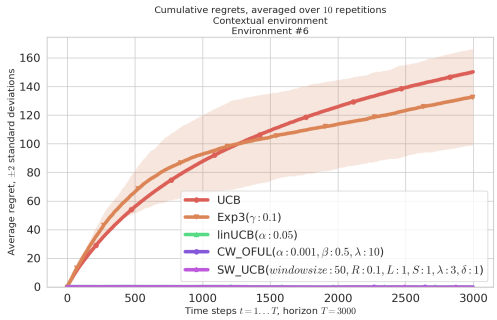

Figure 5: The second pattern in a contextual environment with identical low weights, identical higher context means and strongly varying context variance between every arm

The third pattern seems to occur when the context means are low and the context variance is very different for each arm. In these environments, there is a lot more variation based on θ^* and SW-UCB specifically tends to perform a lot worse, sometimes even to the point of having worse performance at the time horizon of 3000 than one of the stochastic algorithms. All algorithms tend to have relatively high variance between repetitions in these environments. An example of this can be found in Figure 6.

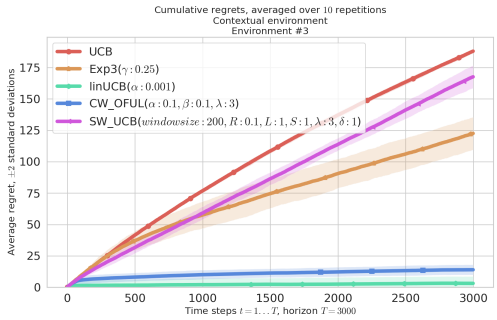

Figure 6: The third pattern in a contextual environment with identical higher weights, identical low context means and identical low context variance

The last pattern occurs when the context means are high and the variance is low. The pattern is similar to the first behaviour (Figure 4) but the cumulative regret of all of the algorithms is so low within the used horizon that the variance is very high compared to the regret and the data is very noisy. The stochastic algorithms appear to behave similar to an arbitrary random algorithm and the contextual algorithms all have very low amounts of regret, but no meaningful trends can be seen in this data beyond that. An example of this pattern can be seen in Figure 7.

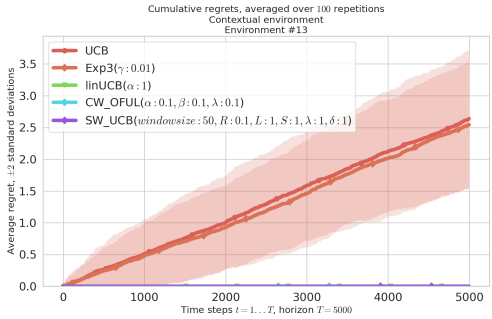

Figure 7: The fourth pattern in a contextual environment with identical higher weights, identical higher context means, identical low context variance, additional repetitions and a higher horizon

When comparing only the contextual algorithms with each other, 30 out of 45 graphs show all algorithms' cumulative regret confidence bounds intersecting, indicating no clear difference in those configurations. In the remaining 15 graphs, SW-UCB always has notably higher regret or a similar value at the time horizon compared to the other two algorithms. In 11 of those graphs, linUCB and CW-OFUL both perform similarly. This leaves 4 notable experiments.
In 2 of them linUCB ends with the highest regret, with SW-UCB and CW-OFUL performing similarly to each other. Both environments have high context variance and means. The weights of the first environment are high, uniform weights and the other has large, negative, uniform weights. An example of this can be seen in Figure 8
In 2 environments linUCB distinctly has the lowest regret at the end of the time horizon. These environments both have low means and strongly varying variances. The weights of one are low and uniform, the other has strongly varying weights. An example of this can be seen in Figure 9
Environments with the same means and variances but different weights show the same patterns as these last four, but the difference in mean final regret is not strong enough for the results of linUCB and CW-OFUL to be outside of each others' confidence bounds.

3.3 Perturbed environments

In perturbed environments, similar patterns seem to hold when comparing stochastic and contextual arguments; The stochastic algorithms generally gather the most regret, with SW-UCB gathering similar amounts to the stochastic environments in a small number of environments and performing similarly to the contextual algorithms in most others.
The first difference is seen when comparing the two stochastic algorithms with each other. While UCB generally still produces the least regret if there is a noteworthy difference,

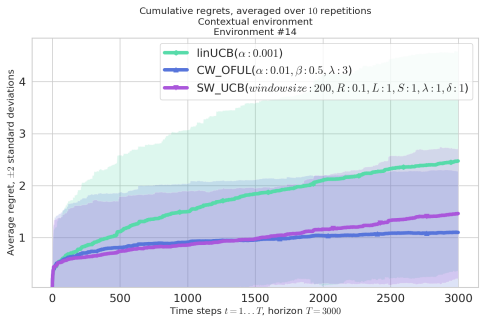

Figure 8: Contextual environment with identical higher weights, identical higher context means and identical higher context variance

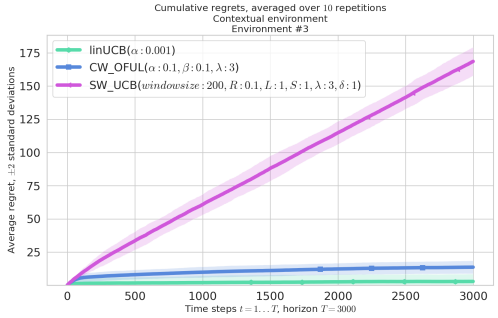

Figure 9: Contextual environment with identical low weights, identical low context means and strongly varying context variance

that occurs less often. In 14 out of 76 tested environments UCB has statistically significantly less regret at the end of the experiment. In one environment, EXP3 actually has a lower mean regret than UCB at the end. There are 5 more environments where EXP3 trends towards lower regret than UCB around the end of the experiment, and 12 more where UCB trends towards lower regret than EXP3, but in these cases the algorithms do not achieve a significant enough difference before the end to cross the 95% confidence bounds, so more computation in these environments may result in more information. The remaining 44 environments show UCB and EXP3 performing very similarly.

There is more to be said when comparing the various contextual algorithms with each other. In 31 out of 76 tested environments, all three algorithms performed similarly. There were 24 environments where linUCB and CW-OFUL performed similarly to each other but SW-UCB gained notably more regret, and there were 16 environments where SW-UCB and CW-OFUL performed similarly to each other and linUCB gained notably more regret. There were no environments where CW-OFUL performed significantly worse than any of the other algorithms.

This means that the environments in this set where the algorithms gain large enough regret values to produce clear graphs and where there is a significant difference between some of the algorithms tend to follow a few patterns. Within these groupings, the biggest difference between individual experiments is generally the slope and variance of the regret scores of the various algorithms, but the shape of each individual algorithm's regret graph will be mostly the same across these environments. An example of these patterns can be seen in Figures 10, 11 and 12.

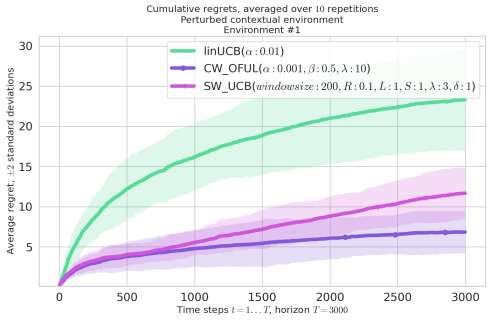

Figure 10: Perturbed contextual environment pattern where linUCB converges far slower than the other two algorithms. SW-UCB and CW-OFUL also generally show some divergence in these experiments

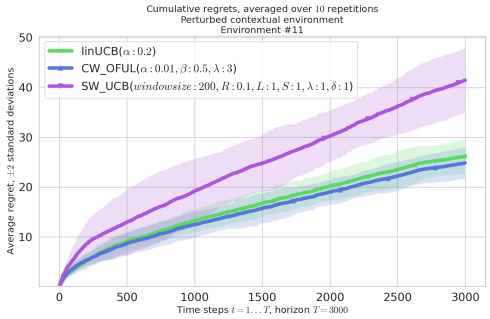

Figure 11: Perturbed contextual environment pattern where SW-UCB trends somewhat steeper after converging than the other two algorithms. LinUCB and CW-OFUL tend to perform very similarly to each other in these environments

The remaining 5 environments showed inconclusive results, they show trends towards certain relations between the algorithms' regret but the confidence bounds are too large to draw any meaningful conclusions.

Of these inconclusive experiments, environments 131, 241 and 311 trended towards an unknown relation between CW-OFUL and SW-UCB's mean regrets and linUCB's mean regret being worse than both others. Environment 451 trended towards an unknown relation between SW-UCB and linUCB's mean regrets and a better mean regret value from CW-OFUL (Figure 13). Environment 651 trended to linUCB's regret being the lowest, CW-OFUL's trending slightly higher, with SW-UCB's being significantly worse (Figure 14).

3.4 Slowly changing environments

In the 27 slowly changing environments similar patterns appear to hold as in stochastic environments and basic

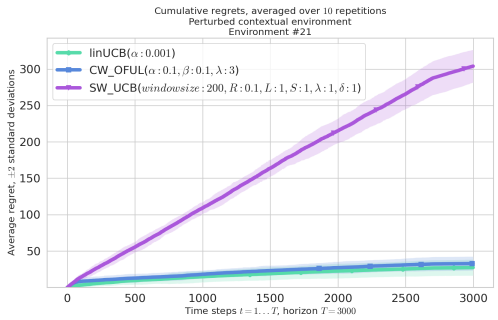

Figure 12: Perturbed contextual environment pattern where SW-UCB trends far steeper than the other two algorithms, with very little variance. LinUCB and CW-OFUL tend to perform very similarly to each other in these environments

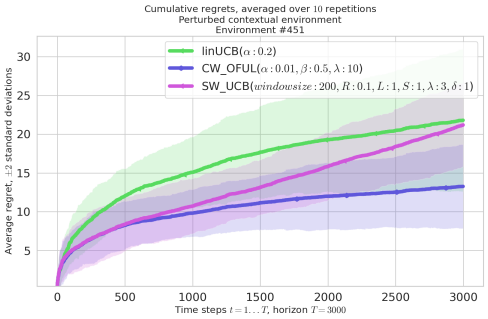

Figure 13: Perturbed contextual environment with identical high weights that get uniformly lowered for a relatively high duration on a relatively large interval. It has identical low context means and identical low variances

contextual environments, the difference remains large, but just like in linear and perturbed contextual environments, in 3 experiments SW-UCB gathers a similarly large amount of regret, such as in Figure 15. A small difference is that in the changing environments this seems to happen when the context means are high and the context variances are low. Aside from the 3 cases where SW-UCB ends with higher average regret than one or both of the stochastic algorithms however, the contextual algorithms gather significantly less regret than the stochastic ones.

Comparing the contextual algorithms to each other, the results of the various environments can once again be separated into 5 groups based on algorithm behaviour.
The first behaviour type (ie. Figure 16) is observed in 5 out of 27 environments and shows linUCB and CW-OFUL with very similar average regret growth, generally very small, and SW-UCB with significantly greater average regret at most timesteps.
The second type of behaviour is observed in 10 environments and shows SW-UCB and CW-OFUL performing very similarly, gathering little regret over time, and linUCB with significantly greater average regret growth over time, such as in Figure 17.

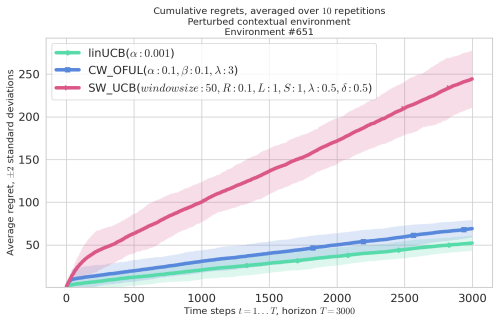

Figure 14: Perturbed contextual environment with strongly varying weights that get uniformly lowered by 2 times the highest weight for a relatively high duration on a relatively large interval. It has identical low context means and identical low context variances

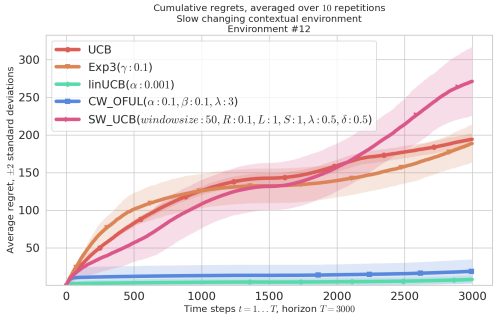

Figure 15: Slowly changing contextual environment with decreasing uniform weights, identical high context means and identical low context variances

The third type of behaviour includes the environments where all of the contextual algorithms perform nearly identically, with high variance relative to the algorithms' mean regrets. This can be observed in 7 of the environments.
The fourth group, consisting of 4 environments, includes the experiments that appear like they would produce noteworthy results at a greater time horizon or with smaller confidence bounds, like for example in Figure 18.

The final type, consisting only of environment 22, shows all of the contextual algorithms with a mean regret of 0.0 and variance of 0 at every time step. The resulting graph can be seen in Figure 19.

4 Responsible Research

This project does not involve data relating to people, so there are no concerns, ethical or otherwise, in that regard.

In terms of repeatability, this paper aims to include as much of the experiment setup and data as possible, including all parameters, results for each test run and the full code base that was used to run these tests.

All of the data obtained in the definitive run of each test is considered, and test runs have only been redone

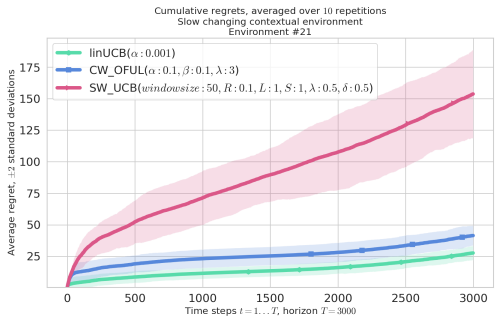

Figure 16: Slowly changing contextual environment with strongly varying weights where the distribution slowly inverts (eg. the highest weight becomes the lowest), identical high context means and identical low context variances

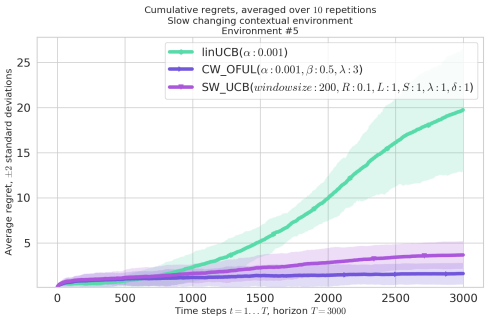

Figure 17: Slowly changing contextual environment with increasing uniform weights, identical high context means and strongly varying context variances

without preserving previous results if the previous run was deemed invalid, due to either a problem with the code, the setup or the graphs.

The full set of randomly generated contexts and rewards, however, is not stored or published. This is not feasible due to the size of the data, which is in the order of hundreds of gigabytes due to the number of tests and repetitions.

5 Discussion and Future Work

As one might expect, the results of this study show that stochastic algorithms nearly always gather far more regret over time than contextual ones in a contextual environment, be it static, perturbed or changing. Our results show UCB generally performing somewhat better than EXP3 in the environments we covered, with few exceptions. This aligns with prior analysis by other studies and can mostly be explained with the extra information the contextual algorithms can use compared to stochastic ones. This does come at the price of a very slight loss in flexibility however, as stochastic algorithms can be applied to contextual environments, but not the other way around.

The performance of the various contextual algorithms is generally very similar in linear contextual environments.

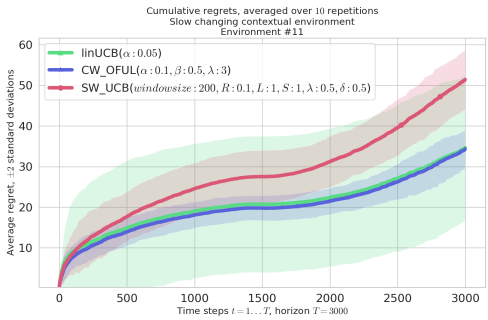

Figure 18: Slowly changing contextual environment with increasing uniform weights, identical high context means and strongly varying context variances

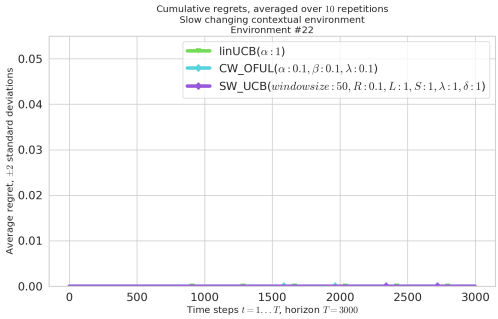

Figure 19: Slowly changing contextual environment with increasing uniform weights, identical high context means and strongly varying context variances

Both linUCB and CW-OFUL seem to be very consistent in their performance, with SW-UCB sometimes performing slightly worse. This may be slightly helped by tuning its parameters, but from our results it seems unlikely that it would outperform linUCB and CW-OFUL consistently enough to justify using it in this environment over one of the others. We cannot give a conclusive answer on which of these two algorithms would definitively be better in static environments however, as their performance in our tests has mostly been within each other's confidence bounds. This means that for a more conclusive answer, further research would be required.

The comparison between contextual environments shifts slightly in perturbed and slowly changing environments however. In both of these setups linUCB and SW-UCB show high regret values in certain, albeit somewhat rare, configurations. This is not unlike the behaviour of SW-UCB we described in linear contextual environments. Interestingly, at least in our tests, these algorithms do seem to show this behaviour in different environments, never in the same one. The cause of these high regrets seems different in contextual, perturbed and changing environments however, and also doesn't appear to be the same for the two algorithms, so we don't have a conclusive reason for this occurrence. CW-OFUL does seem to consistently either be within confidence bounds of the algorithm with the lowest mean regret, or to be that algorithm

itself. Based on this it seems from our results, given that it is feasible to tune its parameters based on the environment it will be implemented in, that CW-OFUL is generally the best choice in both types of changing environments. This conclusion is somewhat surprising, as SW-UCB was specifically designed for environments that gradually change over time, whereas CW-OFUL was designed for adversarial contextual environments.

5.1 Potential future work

While performing this study, a few points of interest came to our attention which may be of value to research, but that were infeasible to cover thoroughly in this paper due to our various limitations.
A logical followup to this study would be to do more reasearch into the cause of the high regrets that linUCB and SW-UCB sometimes show.
Another would be to repeat our tests with more repetitions and a higher time horizon, given more time and/or computation power, as a lot of the experiments ran in this study were inconclusive due to limitations in these resources. In the same vein, it would make sense to gather more robust data on the influence of specific properties of the environment, like the number of arms, dimension of the contexts and the amount of noise, on the performance of the algorithms.
Beyond that, it would be of value to cover more types of environments and bandit algorithms in general, including but not limited to higher order algorithms that estimate their own parameters instead of requiring information about the dataset or tuning (ie. BOB [7]), or adversarial environments [4].
Another option that remains, which was originally intended to be part of this study but that was left out due to a lack of usable data, is using data from user interactions to test bandit algorithms to simulate an environment more representative of what the algorithms might encounter in practice.

6 Conclusions

The aim of this paper was to find the difference in regret-based performance between the MAB algorithms UCB, EXP3, linUCB, CW-OFUL in static, perturbed or slowly changing environments and with or without user data as context vectors.
In the basic, stochastic environments considered in this study, UCB and EXP3 generally performed roughly the same, but in the rare cases that there was a difference, UCB was generally the one that performed better.
In contextual environments, the stochastic algorithms are generally the ones with the worst performance.
LinUCB and CW-OFUL both consistently perform the best in linear contextual environments, but in both of the changing contextual environments in this study, CW-OFUL is the algorithm with the most consistent good performance.
Interestingly, SW-UCB often does not perform as well as linUCB and CW-OFUL in slowly changing environments, even though it has been designed for that.
In conclusion, it appears from our results that UCB is the best performing algorithm of our collection in basic, stochastic environments with normally distributed rewards and CW-OFUL is one of the best performing algorithms in all of the contextual environments we considered, with linUCB being a slightly better static environments, especially if one wants to reduce the need to tune algorithm parameters.

7 Acknowledgements

Finally, acknowledgement should be given to Julia Olkhovskaya, who was of great help explaining several aspects of multi armed bandit problems, algorithms and environments. Furthermore, I offer my sincere thanks to the various anonymous students who reviewed this paper for their feedback. Finally, I am very grateful for the help from my peer group, and especially the help they offered in finding errors in my code and fixing them.

References

[1] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge University Press, 2020.

[2] T. L. Lai, "Adaptive Treatment Allocation and the Multi-Armed Bandit Problem," *The Annals of Statistics*, vol. 15, no. 3, pp. 1091–1114, 1987. DOI: 10.1214/aos/1176350495. [Online]. Available: https://doi.org/10.1214/aos/1176350495.

[3] L. Besson, *SMPyBandits: an Open-Source Research Framework for Single and Multi-Players Multi-Arms Bandits (MAB) Algorithms in Python*, Online at: GitHub.com/SMPyBandits/SMPyBandits, Code at https://github.com/SMPyBandits/SMPyBandits/, documentation at https://smpybandits.github.io/, 2018. [Online]. Available: https://github.com/SMPyBandits/SMPyBandits/.

[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM journal on computing*, vol. 32, no. 1, pp. 48–77, 2002.

[5] W. Chu, L. Li, L. Reyzin, and R. Schapire, "Contextual bandits with linear payoff functions," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 208–214.

[6] J. He, D. Zhou, T. Zhang, and Q. Gu, "Nearly optimal algorithms for linear contextual bandits with adversarial corruptions," *Advances in neural information processing systems*, vol. 35, pp. 34 614–34 625, 2022.

[7] W. C. Cheung, D. Simchi-Levi, and R. Zhu, "Learning to optimize under non-stationarity," in *The 22nd International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, pp. 1079–1087.

[8] P. Auer, "Using Confidence Bounds for Exploitation-Exploration Trade-offs," *Journal of Machine Learning Research*, vol. 3, pp. 397–422, 2002.

[9] N. Abe, A. W. Biermann, and P. M. Long, "Reinforcement Learning with Immediate Rewards and Linear Hypotheses," *Algorithmica*, vol. 37, pp. 263–293, 2003. DOI: 10.1007/s00453-003-1038-1. [Online]. Available: https://doi.org/10.1007/s00453-003-1038-1.

[10] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10, ACM, Apr. 2010. DOI: 10.1145/1772690.1772758. [Online]. Available: http://dx.doi.org/10.1145/1772690.1772758.

[11] D. Arsene, C. M. Boon, M. Herrebout, W. Hu, and R. Owczarski, *Contextual SMPyBandits*, Online at: GitHub.com/thatCbean/SMPyBandits, 2024. [Online]. Available: https://github.com/thatCbean/SMPyBandits/.

[12] C. M. Boon, *Comparisons of bandit algorithms in static and changing systems*, Online at: https://drive.google.com/drive/folders/1JrtXX7Y-DM2ZRKCvpuFMjDQNPYH5rWFE?usp=sharing, 2024. [Online]. Available: https://drive.google.com/drive/folders/1JrtXX7Y-DM2ZRKCvpuFMjDQNPYH5rWFE?usp=sharing.

A Algorithm instance parameters

Table 2: EXP3 Parameter values

γ
0.01
0.05
0.1
0.25
0.5
0.75

Table 3: linUCB Parameter values

α
100.0
50.0
20.0
10.0
5.0
2.0
1.0
0.5
0.2
0.1
0.05
0.01
0.001

Table 4: CW-OFUL Parameter values

α	β	λ
0.1	0.5	0.1
0.01	0.5	0.1
0.1	0.1	0.1
0.01	0.1	0.1
0.1	0.5	0.5
0.01	0.5	0.5
0.1	0.1	3
0.1	0.5	3
0.01	0.5	3
0.01	0.5	10
0.001	0.5	3
0.001	0.5	10

Table 5: SW-UCB Parameter values

$window\ size$	R	λ	δ
50	0.1	1.0	1.0
200	0.1	1.0	1.0
50	0.1	3.0	1.0
200	0.1	3.0	1.0
50	0.4	3.0	1.0
200	0.4	3.0	1.0
50	0.1	3.0	0.5
200	0.1	3.0	0.5
50	0.1	0.5	0.5
200	0.1	0.5	0.5

B Enlarged graphs

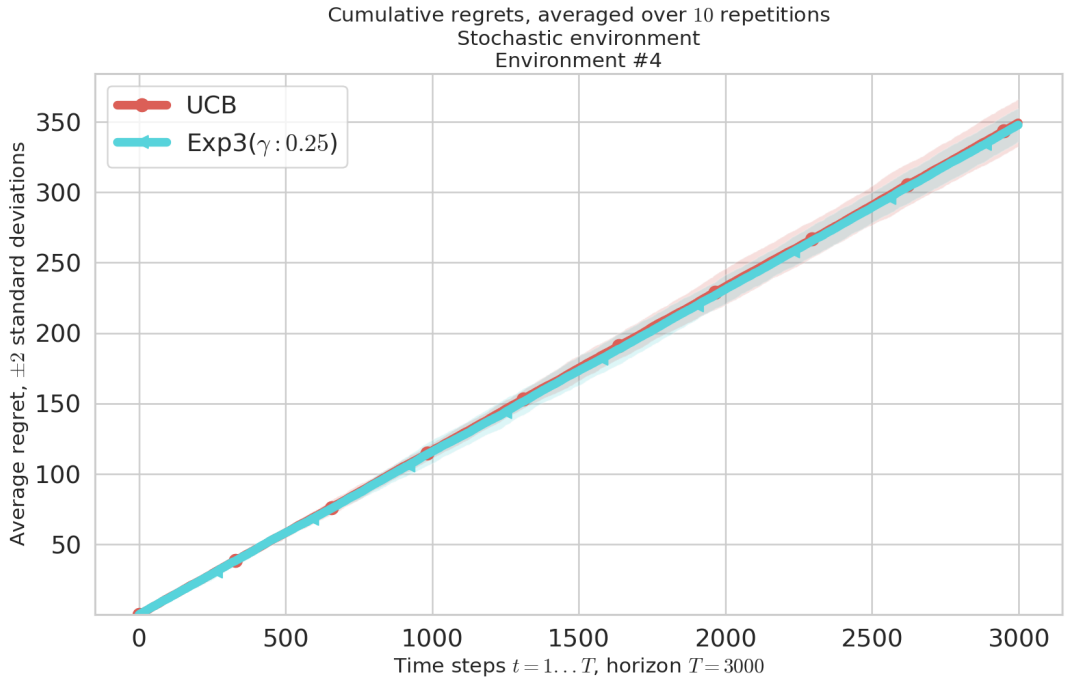

Figure 1: Stochastic environment with identical reward mean and variance for every arm

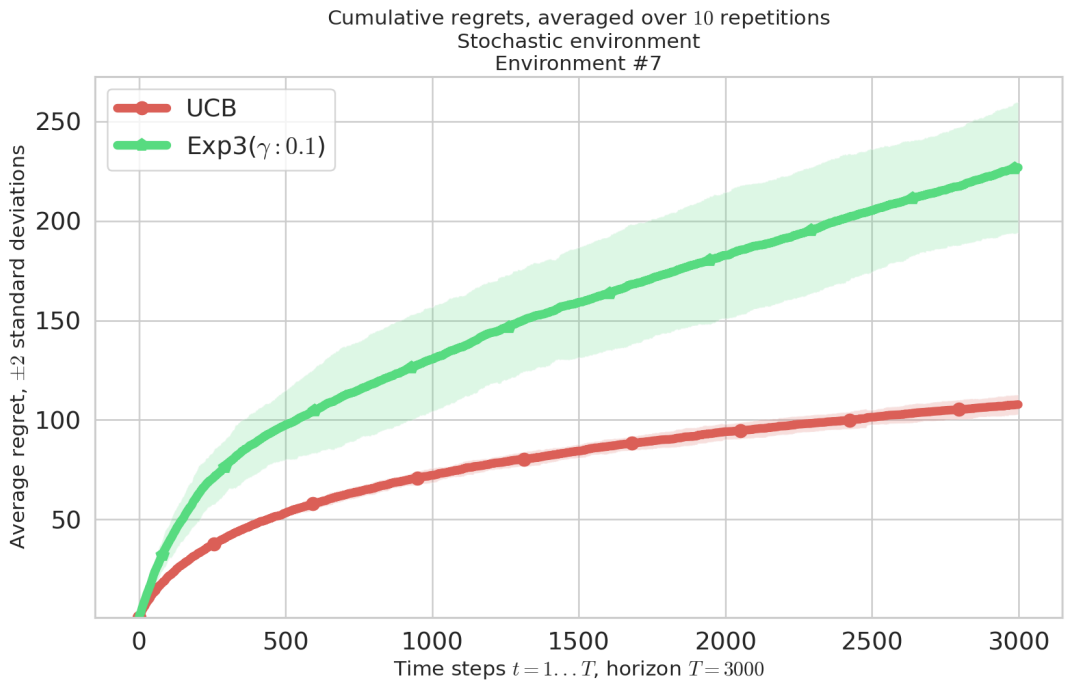

Figure 2: Stochastic environment with different reward mean for every arm and low reward variance

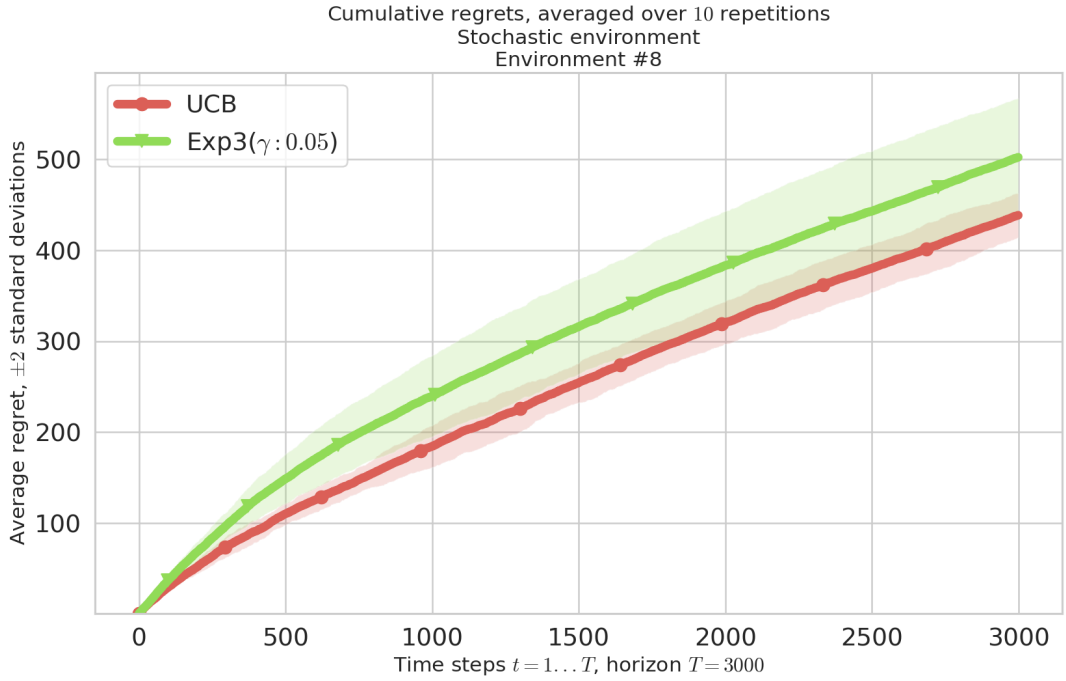

Figure 3: Stochastic environment with different reward mean for every arm and higher reward variance

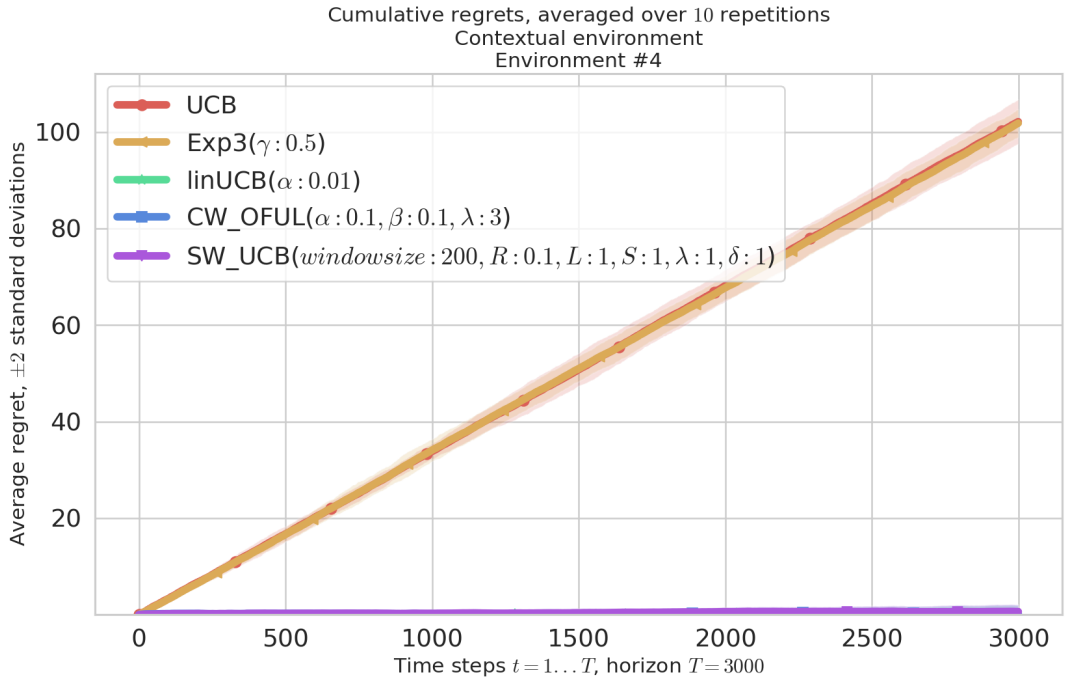

Figure 4: Pattern 1 in a contextual environment with identical low weights, identical low context means and low identical context variance

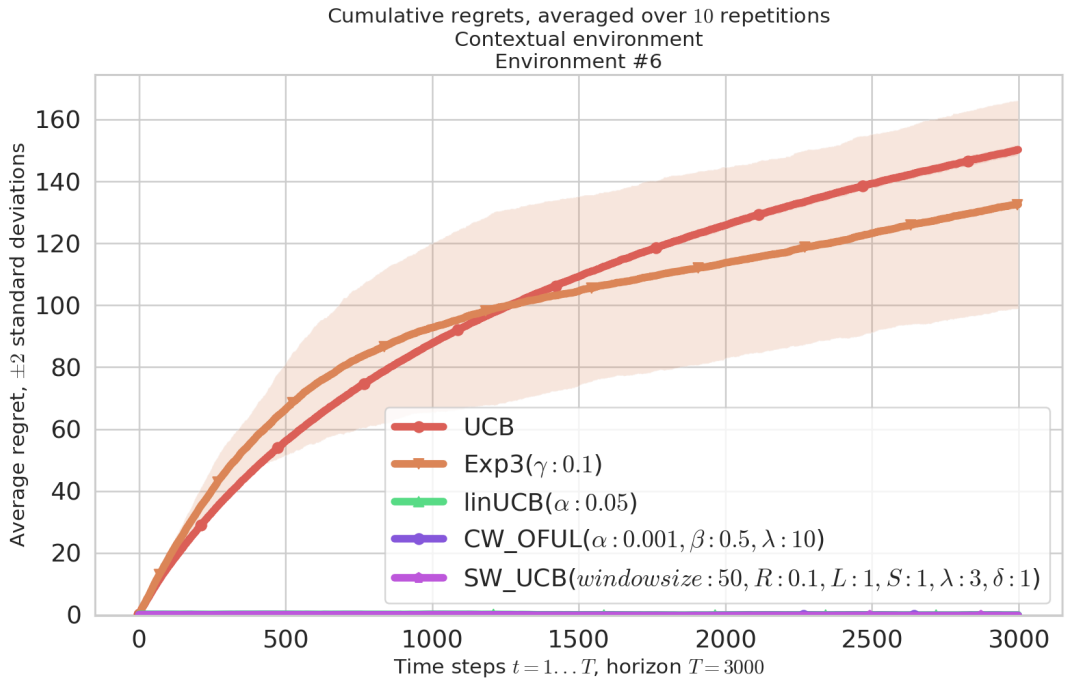

Figure 5: Pattern 2 in a contextual environment with identical low weights, identical higher context means and strongly varying context variance between every arm

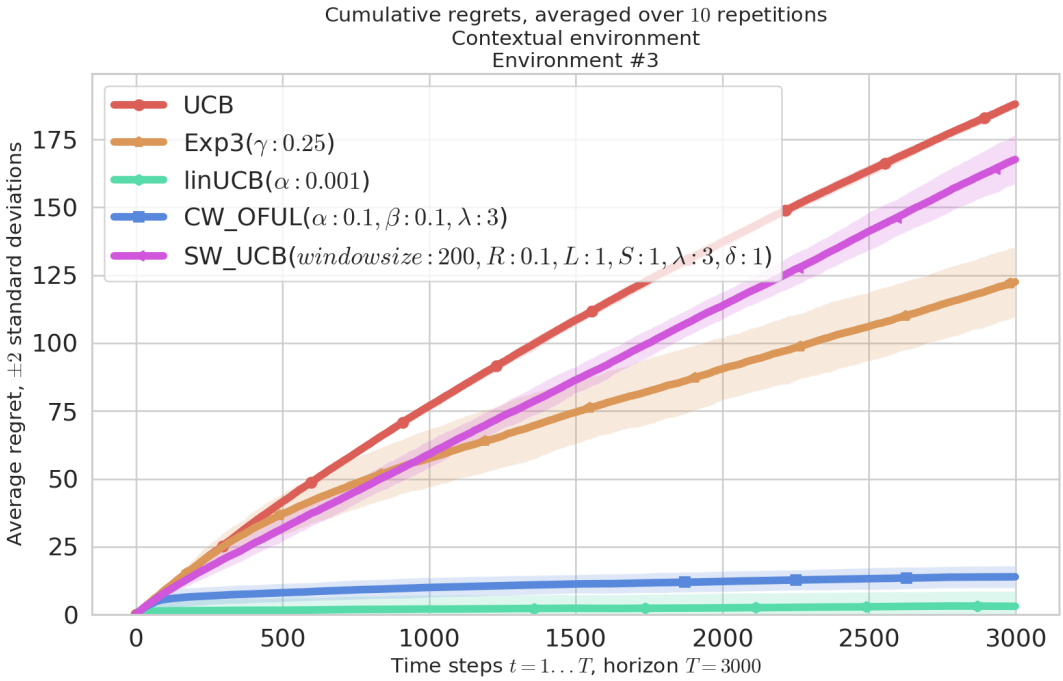

Figure 6: Pattern 3 in a contextual environment with identical higher weights, identical low context means and identical low context variance

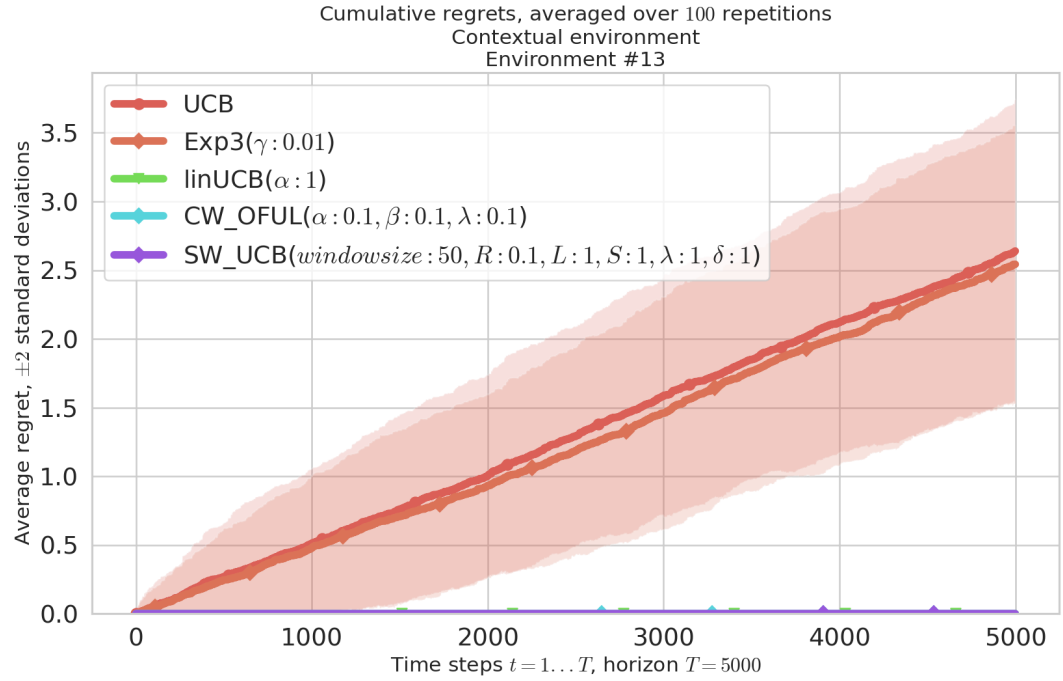

Figure 7: Pattern 4 in a contextual environment with identical higher weights, identical higher context means, identical low context variance, additional repetitions and a higher horizon

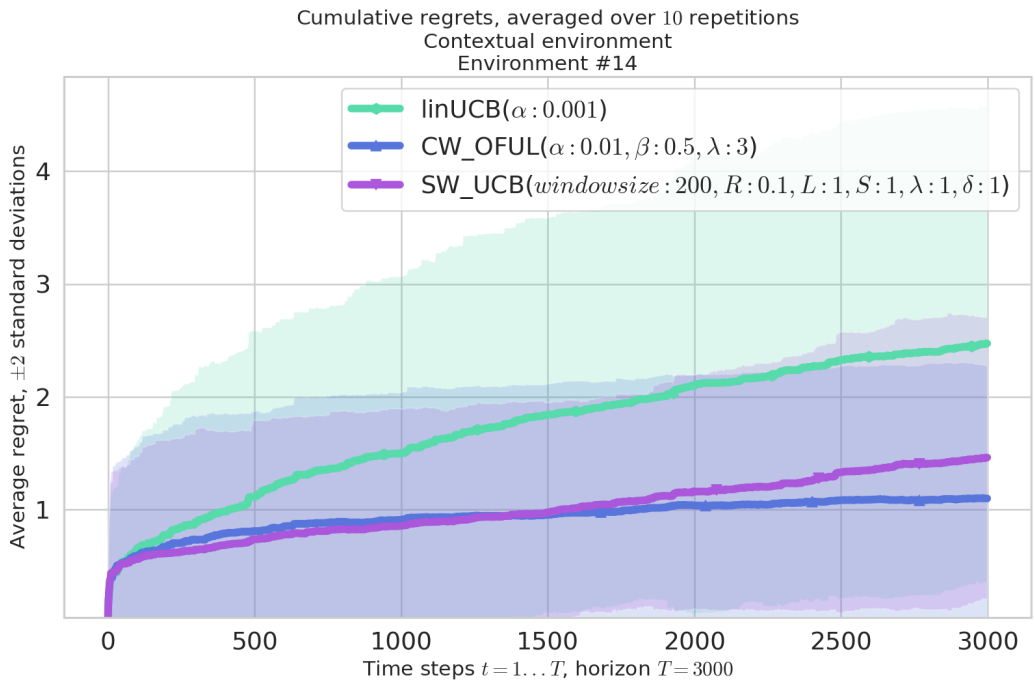

Figure 8: Contextual environment with identical higher weights, identical higher context means and identical higher context variance

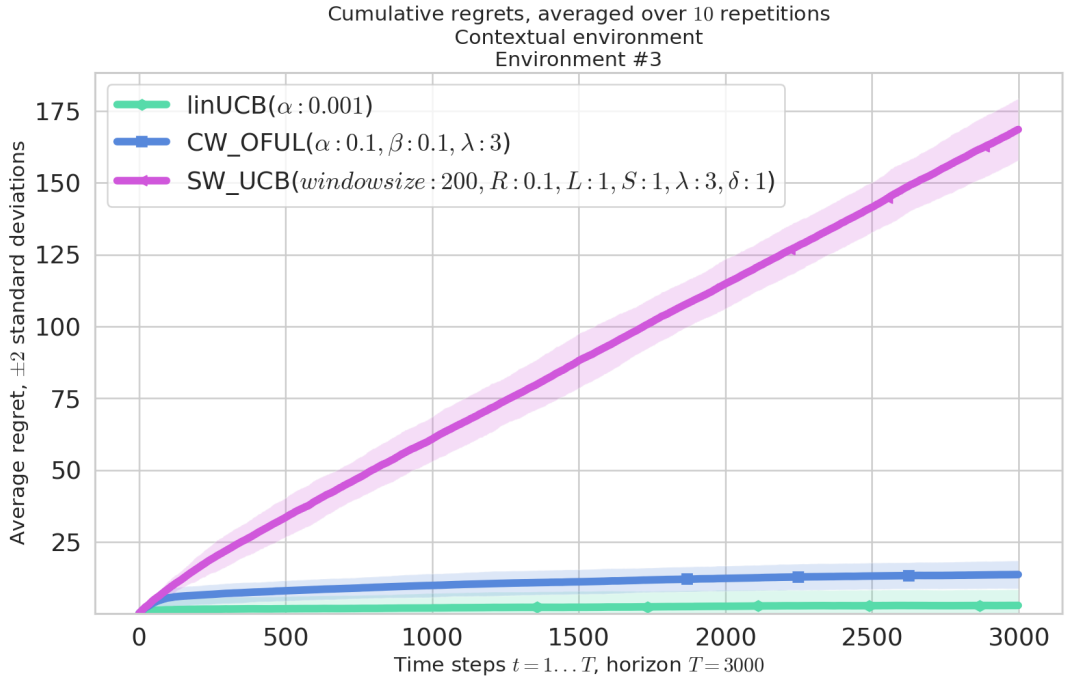

Figure 9: Contextual environment with identical low weights, identical low context means and strongly varying context variance

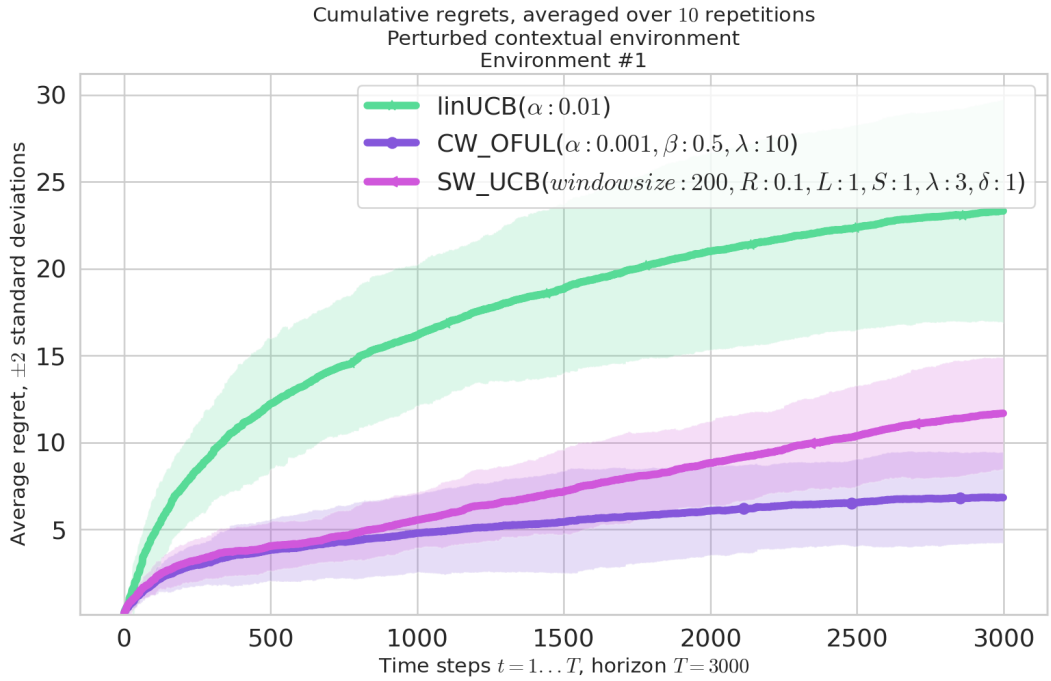

Figure 10: Perturbed contextual environment pattern where linUCB converges far slower than the other two algorithms. SW-UCB and CW-OFUL also generally show some divergence in these experiments

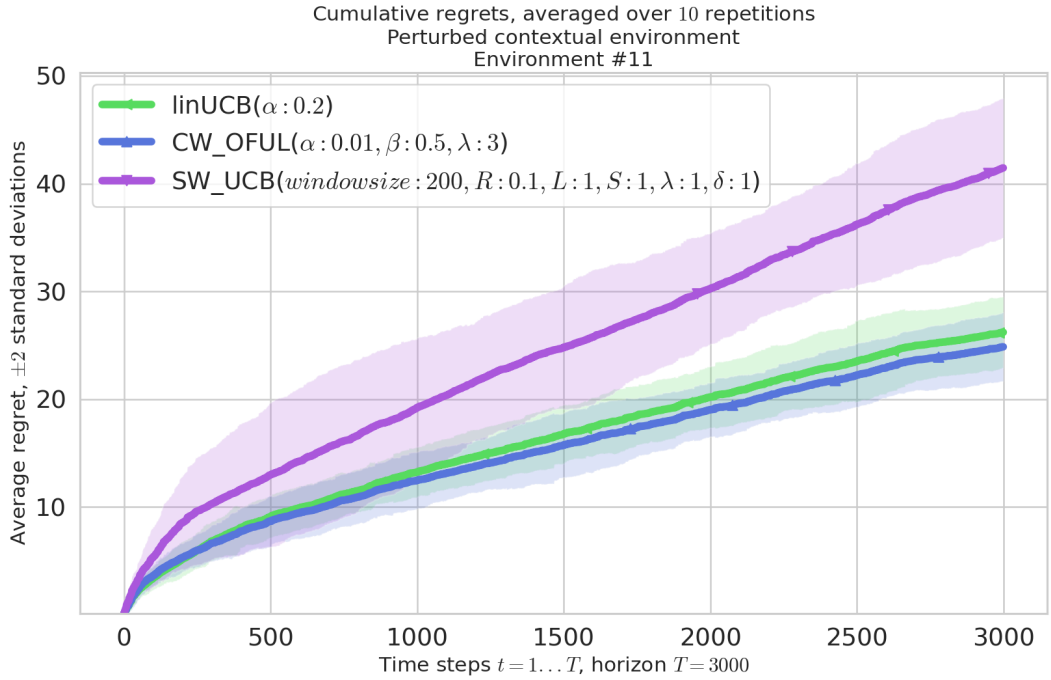

Figure 11: Perturbed contextual environment pattern where SW-UCB trends somewhat steeper after converging than the other two algorithms. LinUCB and CW-OFUL tend to perform very similarly to each other in these environments

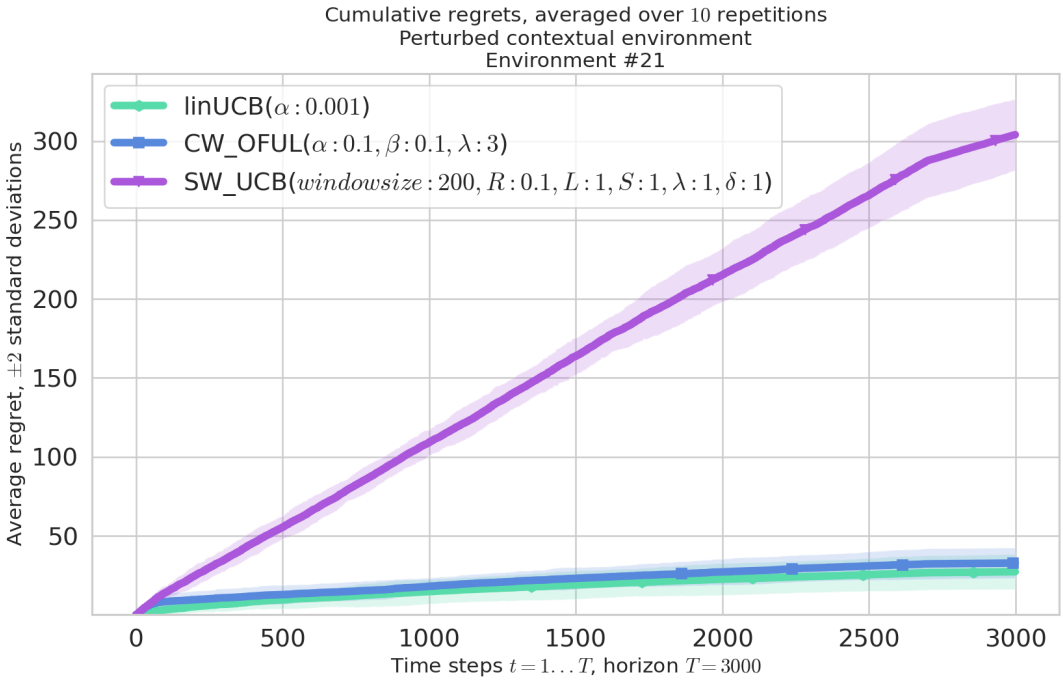

Figure 12: Perturbed contextual environment pattern where SW-UCB trends far steeper than the other two algorithms, with very little variance. LinUCB and CW-OFUL tend to perform very similarly to each other in these environments

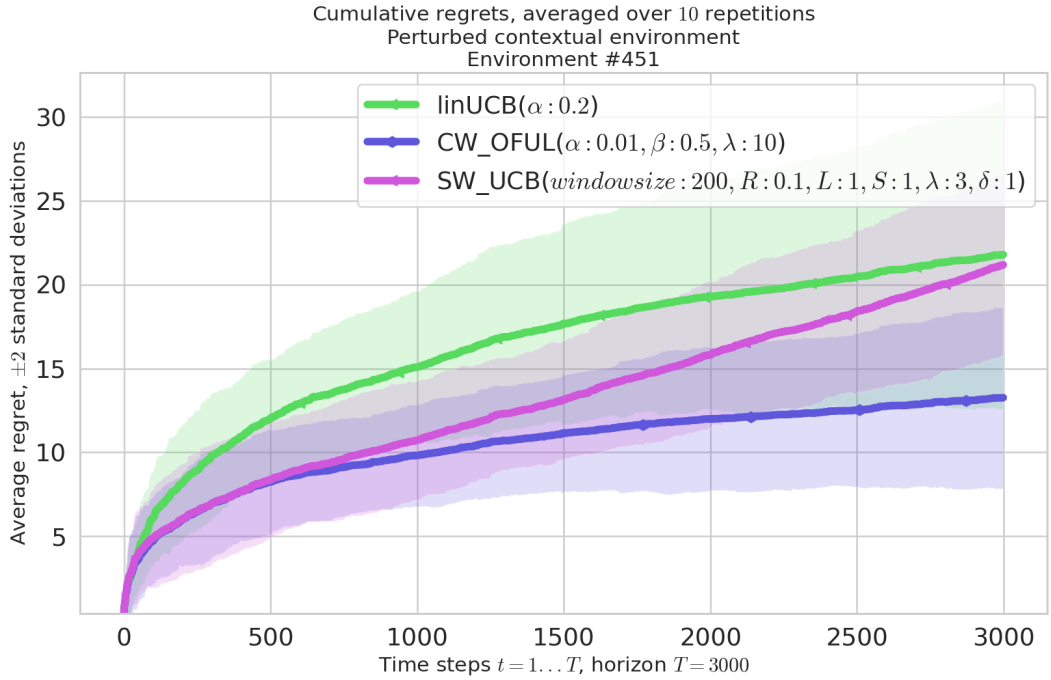

Figure 13: Perturbed contextual environment with identical high weights that get uniformly lowered for a relatively high duration on a relatively large interval. It has identical low context means and identical low variances

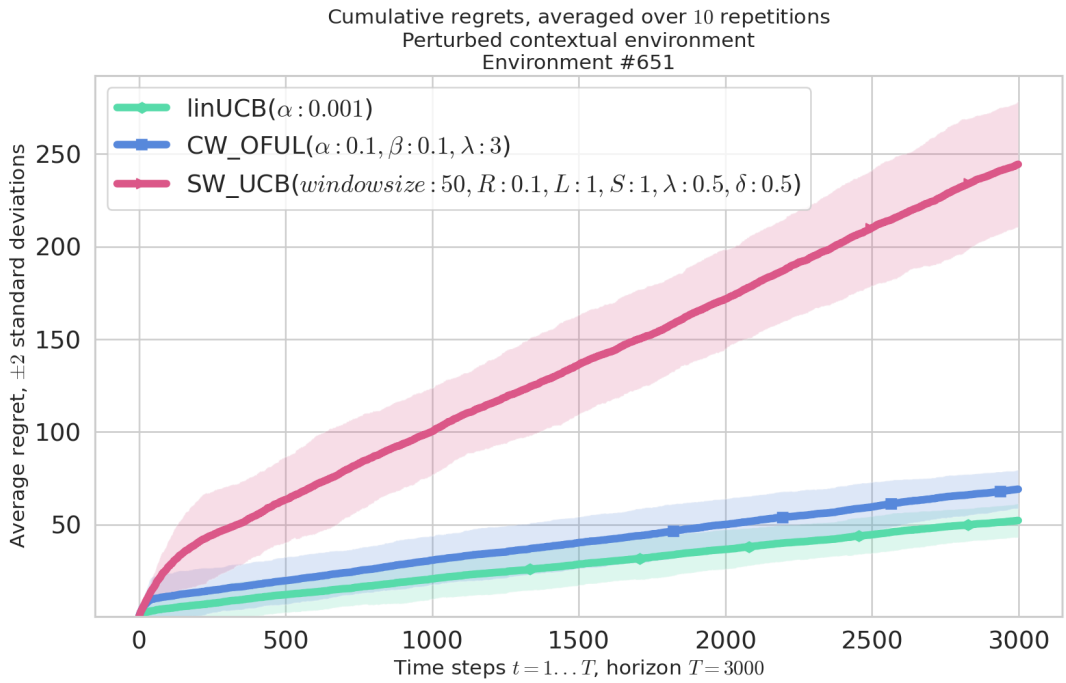

Figure 14: Perturbed contextual environment with strongly varying weights that get uniformly lowered by 2 times the highest weight for a relatively high duration on a relatively large interval. It has identical low context means and identical low context variances

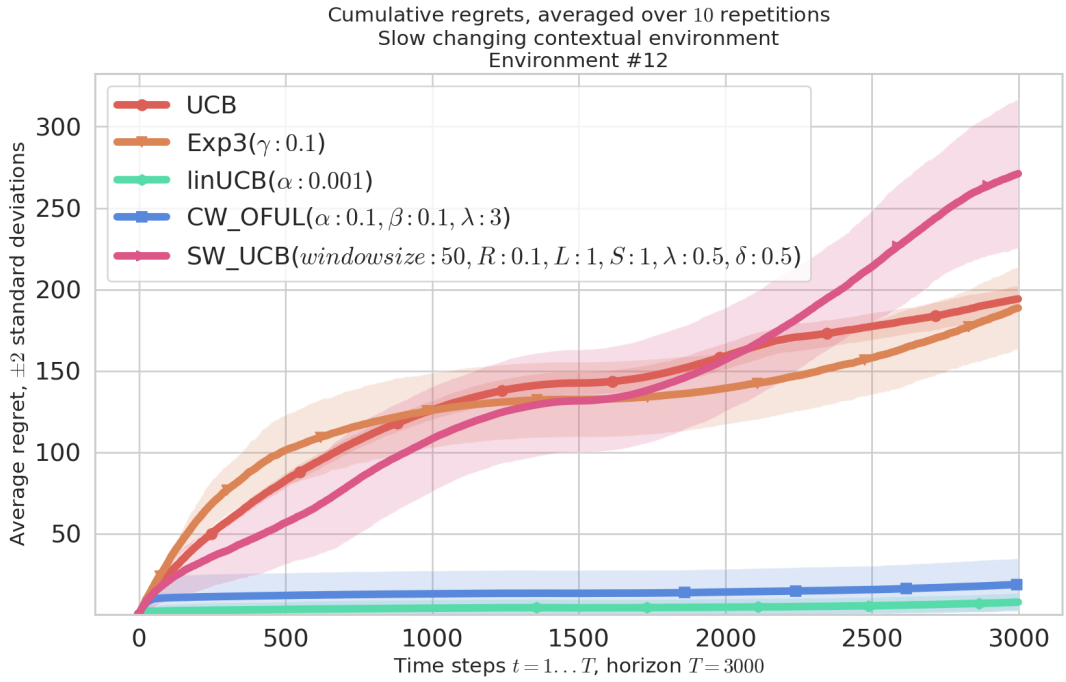

Figure 15: Slowly changing contextual environment with decreasing uniform weights, identical high context means and identical low context variances

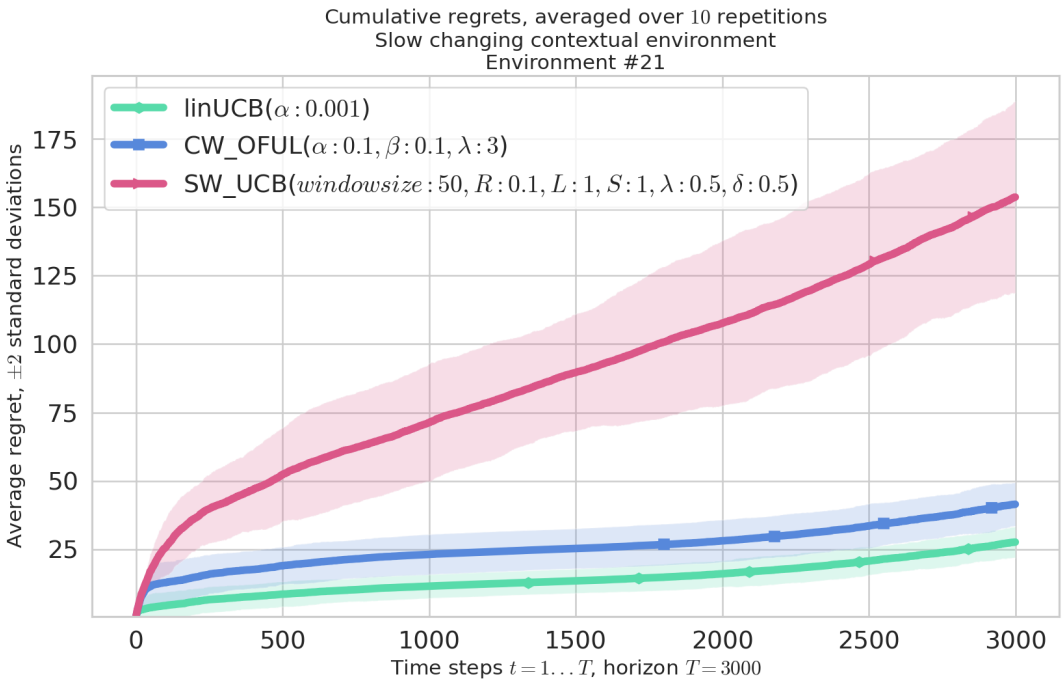

Figure 16: Slowly changing contextual environment with strongly varying weights where the distribution slowly inverts (eg. the highest weight becomes the lowest), identical high context means and identical low context variances

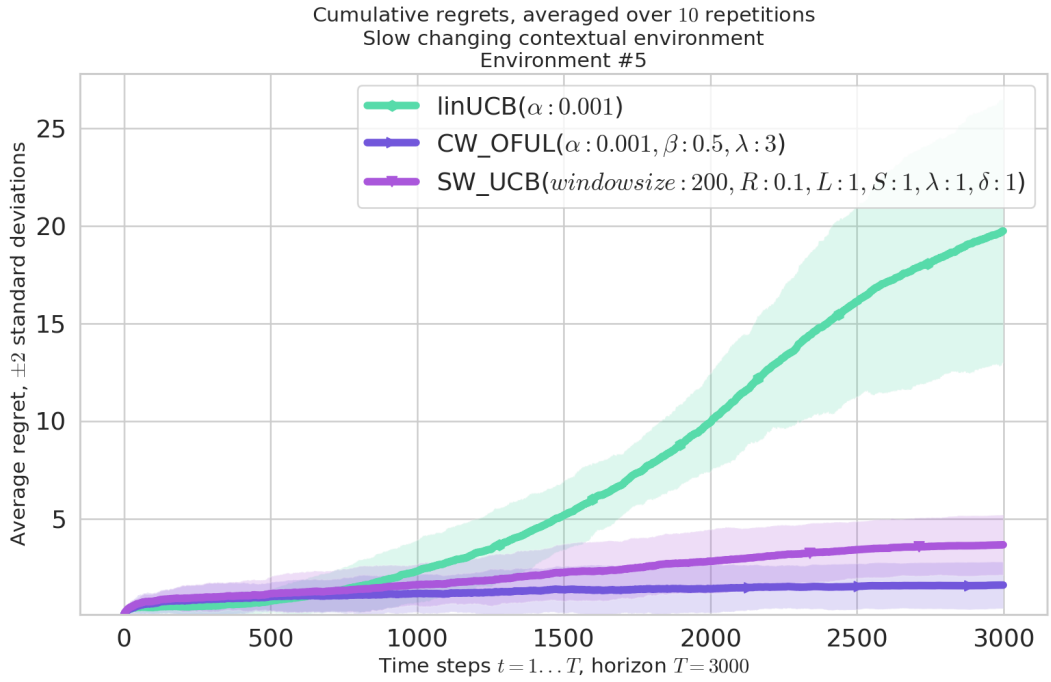

Figure 17: Slowly changing contextual environment with increasing uniform weights, identical high context means and strongly varying context variances

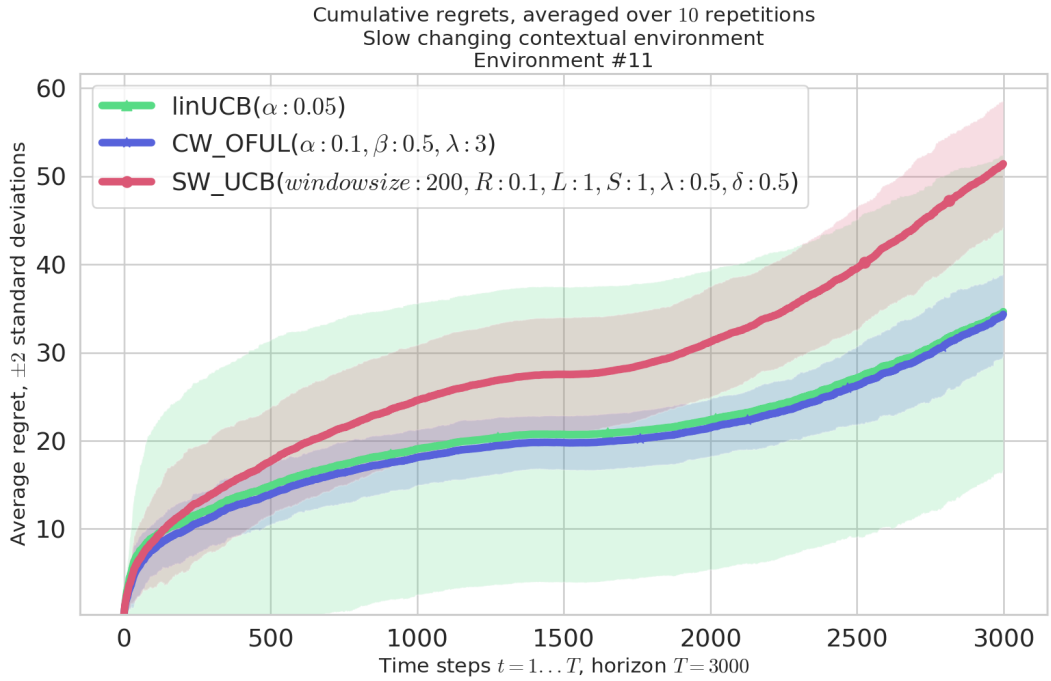

Figure 18: Slowly changing contextual environment with increasing uniform weights, identical high context means and strongly varying context variances

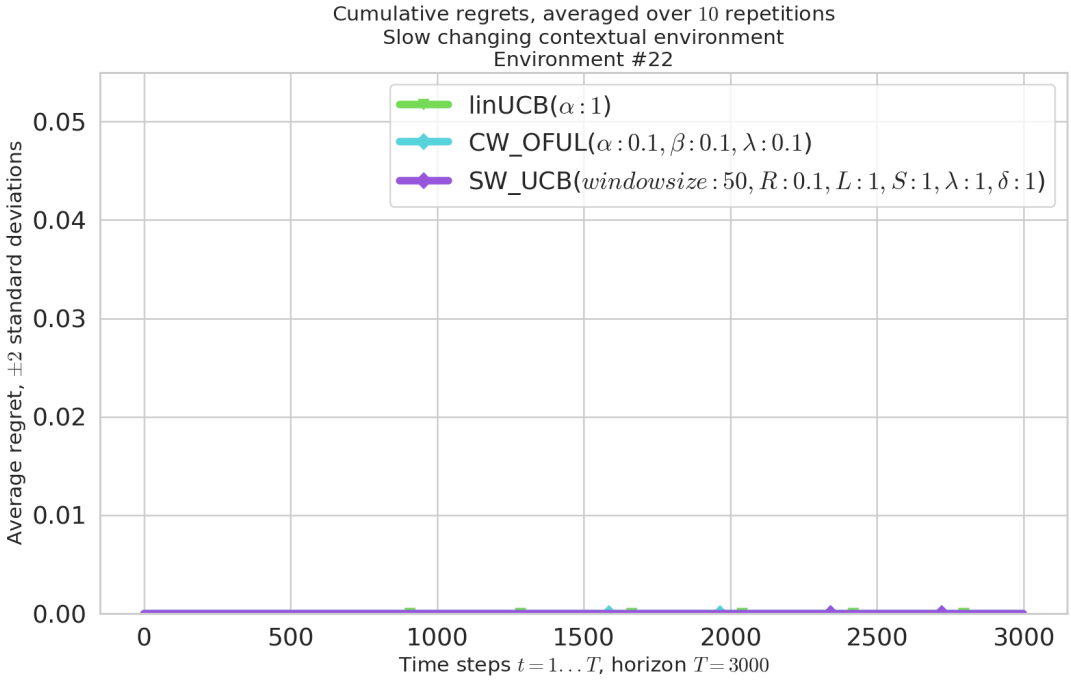

Figure 19: Slowly changing contextual environment with increasing uniform weights, identical high context means and strongly varying context variances