Hierarchical Bayesian Attractor Model for Dynamic Task Allocation in Edge-Cloud Computing

Tatsuya Otoshi[¶], Masayuki Murata[†], Hideyuki Shimonishi[†], Tetsuya Shimokawa[†]

***Abstract*—Edge computing responsive applications have been gaining attention in recent years, especially in AI technology. AI distillation techniques allow compact models to be placed at the edge or terminal, where computational power is limited, with lower latency than can be processed by AI in the cloud. However, AI with smaller models is generally less accurate, so the tradeoff between accuracy and latency, as well as power consumption, must be taken into account when determining processing assignments. Conventionally, such task allocation problems have required heuristic release due to computational difficulties. On the other hand, the heuristic release method has a problem of deviation from the optimal solution when the environment is quasi-static. Our research group takes the approach of continuously searching for the optimal solution in a quasi-static environment while immediately determining the quasi-optimal solution for dynamic environmental changes based on similarities with the past quasi-static environment. In particular, the Bayesian attractor model (BAM), which models brain decision making, is used to select a quasi-optimal solution based on similarity. However, BAM has the problem that appropriate selection becomes difficult when the number of alternatives increases. In this paper, we extend the BAM to a hierarchical model, inspired by the fact that in human decision making, related concepts and operations are grouped into chunks and organized hierarchically. We show that this allows us to maintain a high rate of correct responses even when the number of choices increases. We also investigate the composition of appropriate conceptual hierarchies and updating methods in the temporal hierarchy.**

***Index Terms*—Bayesian Attractor Model, Hierarchical Model, Intention, Edge-Cloud Computing**

I. Introduction

Edge computing, which executes information processing in close to the terminal device has been attracting attention in recent years, especially in AI technologies [1]. For latency-critical applications such as teleoperation, the communication latency to and from the cloud is large with conventional information processing in the cloud [2].

However, it is difficult to run huge AI models as-is in a match because it is difficult to place a large amount of computational resources in the edge compared to the cloud. Using the AI distilling technique [3], it is possible to convert a huge AI model into a small AI model. This makes it possible to run AI processing on computers with scarce computational resources, such as Edge. In addition, application development environments that support both server-side and client-side AI processing, such as Tensorflow.js [4], is also in the works. On the other hand, it is known that smaller models obtained by distillation are less accurate than the original larger models. In addition, power consumption is also important in practical applications, and appropriate placement is necessary to take into account the different power consumption characteristics of different computers. Therefore, it is important to appropriately allocate processing to terminals, edges, and the cloud by considering the trade-off between computational resources, accuracy, and power consumption.

This is a kind of task allocation problem, where how much processing should be allocated to the terminal, edge, and cloud. However, since the processing accuracy and latency requirements vary depending on the real-time status of the application, the task allocation is dynamic, including reallocation. In general, the task allocation problem is a combinatorial optimization problem, which takes time to compute, so it is often assumed to be a static task allocation where the allocation can be computed in advance [5]. Methods such as sequentially allocating arriving tasks or using heuristics to find approximate solutions have been proposed, but they naturally, lead to deviations from the optimal solution. In particular, the problem with simply finding an approximate the solution is that once the application-derived fluctuations settle down and a quasi-static environment arises, the settings that deviate from the optimal solution will continue to be used. Thus, it is important to maintain near-optimal settings in quasi-static situations while also responding to changes in dynamic situations.

Our research group proposes an approach to solving such dynamic assignments by combining short-term and long-term decision making. That is, quick decisions for dynamic changes in the situation are made by short-term decision making, while optimal solutions for a quasi-static environment are computed in the background by long-term decision making. These long-term and short-term decisions work together to solve the problem. The solution obtained by the long-term decision-making is given as a candidate solution for the short-term decision-making, and the short-term decision-making selects a solution in the past quasi-static environment similar to the current situation. For long-term decisions, conventional static optimization can be used because time-consuming computation is acceptable.

Therefore, this paper focuses on short-term decisions with given solutions in a quasi-static environment. In order to judge the similarity of situations in short-term decision making, it is necessary to be resistant to the effects of noise, such as temporary fluctuations in observed values. Therefore, the Bayesian attractor model (BAM) [6], which models the brain's decision making using noisy time series as input, is used to realize

¶ Graduate School of Economics, Osaka University
†Graduate School of Information Science and Technology, Osaka University

decisions that can withstand noise in real environments.

Our research group has extended the Bayesian attractor model to cope with unknown situations, but we have had problems with appropriate cognition when the number of attractors increases. On the other hand, people are always making decisions from a huge number of alternatives. Human decision making has a hierarchical structure, and it is said that this hierarchical structure helps speed up decision making. In other words, a series of related actions are grouped together as a single chunk, and decisions are made on the chunks.

This paper introduces a hierarchical structure to the Bayesian attractor model, and achieves hierarchical decision making by switching attractor sets that summarize attractors. By dividing the model into attractor sets, the number of attractors per set can be kept small, thus eliminating the problem of cognition when the number of attractors increases. We also show that the time required for recognition can be reduced by utilizing structure, such as by grouping similar attractors together. Furthermore, we will compare the case where multiple attractor sets are searched simultaneously and the case where one attractor set is searched intensively, and investigate the appropriate attractor set search method.

The rest of this paper is organized as follows: Section II discusses related research on hierarchical decision models; Section III details the HBAM proposed in this paper; Section IV defines the problem of task assignment in edge cloud computing as envisioned in this paper, and Section V presents an evaluation of HBAM and task assignment using HBAM. Finally, in Section VI we summarize and discuss future work.

II. Related Work

Humans are known to make fast decisions by hierarchizing their behavior. This is called chunking. Chunking shortcuts thinking by putting together a single action that combines multiple primitive actions. This higher-level abstract action is often referred to as an intention. Intention is not only a hierarchization of action space, but also a hierarchization of time direction. Thus, even if actions are switched in a certain continuous time, the intention itself remains the same. Thus, temporal continuity and intention as a set of actions are considered as one of the hierarchies in human decision making.

There are several studies that model intentions in decision making. One literature [7] models intention as an excitation-inhibition relationship between nodes of a hierarchical neural network. In this model, intentions are inhibitory to each other, and when one intention is activated, the corresponding action is activated. However, since the choice of which intention is selected is given as an input to the model, it does not have an automatic intention generation mechanism.

Previous literatures [8], [9] model hierarchical decision making using a hierarchical Gaussian filter. In this model, the upper layers correspond to the situation and differ slightly from the intention of summarizing the behavior. The upper layers in this model determine the variance for the states that the lower layers estimate. When a change in situation occurs, the upper layer estimates a larger variance, which in turn makes it easier for the lower layer to change the current state. Since these estimations are done automatically based on the observed values, the behavior of the upper layer changes spontaneously according to the situation.

Some model attention as an upper layer of decision making [10]. In this model, the variance of the observation is small for the target of attention and large for the target of non-attention. When the variance of the observations is small, the causes in Bayesian estimation are larger, and the state is more likely to be updated under the influence of the observed values. Therefore, the decision-making process changes depending on the presence or absence of attention. Attention mechanisms have recently been the focus of attention in deep learning, as exemplified by the transformer [11]. The difference between attention and intention is whether or not they are connected to the chunking of actions. In the case of attention, it is linked to a specific object of observation rather than a specific action, and does not involve the chunking of cognitive states as in the case of intention.

In addition, the free energy principle [12] has attracted attention as a theoretical study, and the hierarchical structure is said to have an important role there as well. Similar to the predictive coding theory, the free energy principle assumes that the upper layers of the brain predict future sensory stimuli and dynamically changes the gain from the lower to the upper layers according to the error from the prediction.

These are not studies that implemented a mechanism to automatically determine intentions based on observations and make dynamic, hierarchical decisions based on intentions. In this paper, we extend the Bayesian attractor model, which is a dynamic decision-making model, to a model that includes hierarchical intentions.

III. Hierarchical Bayesian Attractor Model (HBAM)

This chapter introduces the Hierarchical Bayesian Attractor Model (HBAM) proposed in this paper, which incorporates the hierarchy of intentions chunking attractors in the traditional Bayesian Attractor Model (BAM) to make decisions based on dynamic time series inputs. This document first provides an overview of the conventional BAM and then presents the problems that arise when the number of attractors in the BAM increases. As a solution to this problem, we introduce a hierarchical structure to BAM and propose a new extended model, HBAM. In addition, the inference method of HBAM and the design method of the hierarchical space will be discussed.

A. Bayesian Attractor Model

BAM is a model of brain decision making [13]. The model consists of an observed value x_t, an internal state z_t and an attractor ϕ_i. The attractor holds several different alternatives tied to a representative value of the observed value μ_i and is embedded in the space of internal states. Decisions are made by updating the internal state based on observed values and its internal identity to a particular attractor. State updating is done by Bayesian estimation, where the posterior distribution of the

internal state is updated according to the observed values and the generative model.

The generative model gives the likelihood of the observed values based on the dynamics of the internal state and the association between the attractor and the representative value. The dynamics of the internal state is given by the Hopfield dynamics f, which embeds the dynamical system with the attractor as a fixed point in the internal state space. This is given by

$$\boldsymbol{z_t} = f(\boldsymbol{z_{t-1}}) + q\boldsymbol{w_t} \tag{1}$$

where $\boldsymbol{w_t}$ is the noise generated in the internal state at each time, and q represents its magnitude and is called dynamic uncertainty. The association between the attractor and the representative value is represented by a matrix $M = (\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_K})$ where each column is the representative value, and the current observed value is predicted from the internal state as follows.

$$\boldsymbol{x_t} = M\sigma(\boldsymbol{z_t}) + s\boldsymbol{v_t} \tag{2}$$

where $\sigma(x)$ is a sigmoid function and maps to a one-hot vector representing index i when $\boldsymbol{z}_t = \phi_i$. The $\boldsymbol{v_t}$ is the noise in the prediction of the observed values at each time point, and s represents its magnitude and is called the sensory uncertainty. Since the model includes noise, the internal state and predictions can be sequentially revised and estimated even when observation errors occur.

However, there is a problem that estimation becomes difficult when the number of attractors increases. Even when the number of attractors increases, the estimation accuracy can be improved by tuning parameters of the model such as the sigmoid slope and s, q, but the tuning itself becomes more severe with the number of attractors. Fig. 1 shows the number of searches when the number of attractors is increased and the parameters are tuned by random search such that the correctness rate of inference exceeds 95%. From the figure, it can be seen that the number of searches for appropriate parameters increases as the number of posterior attractors increases.

These problems are thought to be due to the flat embedding process, which inherently ignores hierarchical structures such as similarity between attractors. Therefore, in this paper, we extend the model to a hierarchical model in which similar attractors are grouped together to form chunks.

B. Hierarchical Extension of BAM

Inspired by the fact that people chunk actions as intentions, we propose a model that makes decisions by switching attractor sets that aggregate attractors. While having a huge number of attractors as a whole, at a certain point in time, the search efficiency is improved by exploring a small number of attractors tied to a specific attractor set. Then, if there is no suitable attractor in the attractor set, it spontaneously switches to another attractor set.

An attractor set is defined as a subset of all K attractors. The set of attractors A is divided into k subsets $A_1, \cdots, A_k$, where each A_I represents one attractor set. Here, it is assumed that

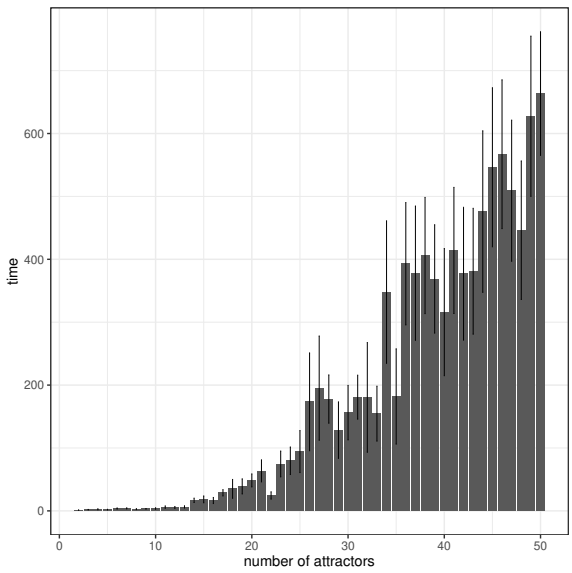


Fig. 1. Number of Attractors and Parameter Tuning Difficulties

the attractor sets do not have overlapping common attractors. That is, the attractor set A_I satisfies the following.

$$A = \bigcup_I A_I \tag{3}$$

$$A_I \subset A \tag{4}$$

$$A_I \cap A_J = \phi,\ (I \neq J) \tag{5}$$

Here, the division of A_I is assumed to be given, and the specific division method for a given set of attractors A will be discussed later.

The HBAM consists of spontaneous switching of attractor sets and a BAM based on each attractor set. Here, the attractor sets are assumed to switch spontaneously with probability p, and the attractor sets after switching follow a multinomial distribution. Thus, the generative model of the HBAM is as follows

$$I_t = \begin{cases} I_{t-1}\ with\ probability\ 1-p \\ I \sim Mult(\boldsymbol{\alpha})\ with\ probability\ p \end{cases} \tag{6}$$

$$\boldsymbol{z}_t^{I_t} = f(\boldsymbol{z}_{t-1}^{I_t}) + q\boldsymbol{w}_t \tag{7}$$

$$\boldsymbol{x}_t = M^I\sigma(\boldsymbol{z}_t^I) + s\boldsymbol{v}_t \tag{8}$$

where I_t represents the index of the active attractor set at time t and $Mult$ represents the Multinomial trial. Although $\boldsymbol{\alpha}$ is a parameter of the distribution and represents the bias of the attractor set chosen, we use $\boldsymbol{\alpha}$, which is uniform in this paper. $\boldsymbol{z}_t^I, M^I$ denotes the internal state $\boldsymbol{z}_t$ and the representative value matrix M in the Ith attractor set.

C. Inference with HBAM

Once HBAM obtains the observed values at each time, it updates the posterior distribution of the internal state $\boldsymbol{z_t^{I_t}}$ and the attractor set index I_t using the generative model described above. In this paper, Particle Filter [14] is used as the algorithm for updating the posterior distribution.

Particle Filter is a sampling-based algorithm that represents the distribution by a large number of sample points (particles). At each time point, it updates the particles according to a generative model using a prior distribution of particles that holds the previous results, and resamples the particles based on the likelihood of the particles under the observed values. The trade-off between accuracy of distribution approximation and computational load can be adjusted depending on the number of particles, and the number of particles used in this paper is 1000.

In this paper, we consider the following two methods for updating the posterior distribution.

- Parallel Update: distribution update without constraints
- Serial Update: distribution update with constraint where $P(I_t)$ is one hot

Parallel Update represents ordinary Bayesian estimation, where I_t is also estimated as a probability distribution. In this case, the model is a mixture of BAMs with different sets of attractors, so the BAMs behave as if they operate in parallel.

Serial Update updates the posterior distribution with the constraint that the probability of a particular I_t is 1. In this case, at each time point, one set of attractors is always active and the state is updated.

For Parallel Update and Serial Update, we evaluate their properties in Section V.

IV. Task Allocation in Edge-Cloud Computing

This paper describes the task assignment problem in edge cloud computing assumed in this paper and its solution by HBAM.

A. Task and AI Model Allocation

In this paper, it is assumed that AI processing, such as image processing, is performed on data such as camera images sensed by the terminal. The user is free to perform AI processing anywhere, but the system can impose restrictions on the end-to-end delay required for processing and the accuracy of the raw AI processing. On the other hand, the system is supposed to satisfy the user's requirements while optimizing the system's objective function, such as power consumption.

The AI that performs the processing shall use models with different sizes depending on the distillation technique. In general, the smaller the model size, the lower the accuracy of the processing, and there is a trade-off between model size and accuracy. The AI uses different computational resources depending on the model size, and the decision variable is where to place the AI in the terminal, edge, or cloud with different resource sizes.

Users using the system are managed on a session-by-session basis, and the decision variable is also where to assign each session to a terminal, an edge, or the cloud.

The above control is an AI and session allocation problem that optimizes the objective function of the system with user requirements as constraints. This involves combinatorial optimization, which requires high computational cost. On the other hand, it requires responsiveness because it involves dynamic changes in the environment, such as changes in user requirements.

Task assignment in such dynamic environment variation is generally done using heuristics, but deviations from the optimal solution are inevitable [15]–[17]. In particular, deviations from the optimal solution become problematic when the environment is quasi-static, such as when each user makes similar requests as a result of using the same application. In a quasi-static environment, there is a risk of sticking to the approximate solution by heuristic, even though the optimal solution can be computed at inherently high computational cost.

B. Selecting Quasi-Static Solution by HBAM

The paper approaches the problem of responsiveness in a fluctuating environment and deviation from the optimal solution in a quasi-static environment by combining short-term and long-term decision making. In a quasi-static environment, the optimal solution is computed by performing optimization in the long term, and in a fluctuating environment, a similar quasi-static environment is identified by HBAM, and the corresponding optimal solution is fed into the system. This allows the system to continuously search for optimal solutions in quasi-static environments while maintaining high responsiveness.

When performing optimization in a quasi-static environment, the amount of available computing resources and user requirements are stored as representative values $\boldsymbol{\mu}_i$, and the optimal solution obtained is stored as $\boldsymbol{\phi}_i$. This is used as an attractor in HBAM, and attractors with similar representative values are grouped together by clustering to form an attractor set.

In a fluctuating environment, HBAM acquires each available amount of computing resources and user requirements as observed values $\boldsymbol{x}_i$ and updates the internal state $I_t, \boldsymbol{z}_t^{I_t}$. Then, the attractor selected according to the internal state is put into the system to achieve high responsiveness.

V. Evaluation

Due to page constraints, the paper only evaluates the effect of introducing a hierarchical structure for HBAM.

Numerical simulations are used to evaluate the basic properties of the HBAM. We also show that HBAM maintains high cognitive accuracy by introducing a hierarchical structure even when the number of tracts as a whole increases compared to the original BAM. Furthermore, we will investigate how the computation time and day time change depending on the granularity at which the attractor set is divided and how the state is updated.

1) Simulation Setting: In this evaluation, the representative values of artificially generated attractors are used to evaluate the behavior of HBAM. Here, the representative value of attractor i is assumed to be $\mu_i = (i, i+1, i+2)$. The number of total attractors should be changed according to the purpose of the simulation. Depending on the number of attractors and the number of attractor sets determined, the attractor sets are divided into attractor sets that group together representative

values that are closer by clustering. The parameters of the HBAM were set to $p = 0.2, q = 1, s = 3$.

The observed values were given 150 time slots and changed every 50 time slots to evaluate whether they could keep up with the changing situation. Because we are interested in switching attractor sets, we give the observed values so that the correct attractor set switches every 50 time slots. Specifically, the first 50 time slots and the last 50 time slots are given observation values corresponding to attractors in the same attractor set. And the middle 50 time slots give the observed values corresponding to attractors in a different attractor set. The percentage of time slots in which the decision state and the correct attractor coincide out of the 150 time slots is the correct response rate.

A comparison with the original BAM is also made, but this time the BAM is assumed to have the same attractors as the HBAM and one attractor set with a flattened hierarchy.

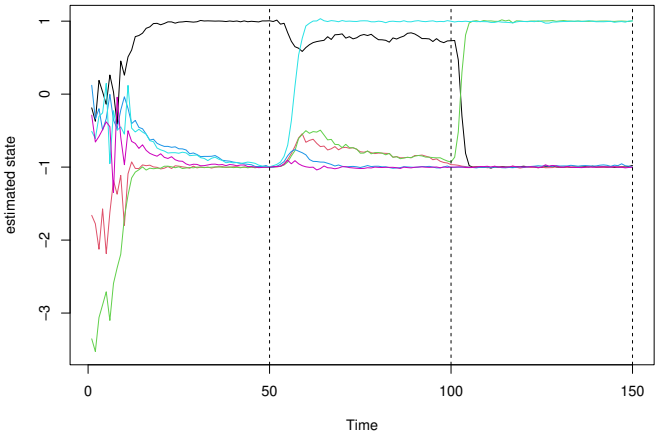
2) Decision State Time Series: As a basic behavior of HBAM, an example of the state update result is shown in Fig. 2. Here, six attractors are split into two sets with three attractors. The two figures are the result of parallel and serial updates, respectively. Each color in the figures indicates the value of the decision state of a different attractor. Here, the attractor corresponding to the correct answer is given an observation value that is black in the first 50 time slots, light blue in the next maintenance time slot, and green in the last 50 time slots.

It can be seen that both series and serial updates are able to change the decision-making state according to the change in observed values every 50 time slots. The speed of the change tends to be slightly faster in the parallel update. This is because in parallel updating, all attractor sets are updated with some degree of activity. This is thought to make the response to changes every 50 time slots that require attractor set switching faster than in series updating.

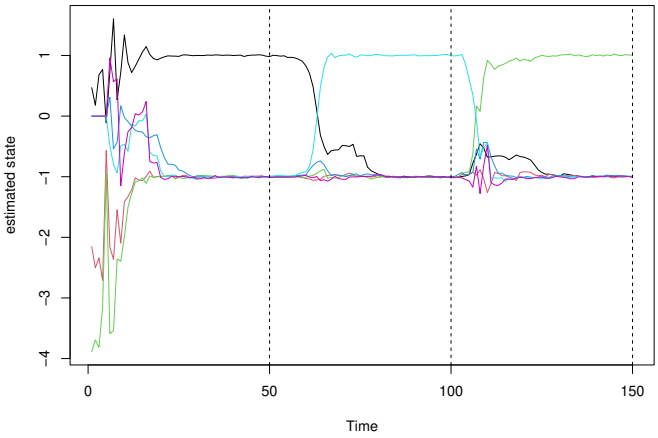
In the case of parallel updates, black and light blue are higher in the 50 to 100 time slots, and both light blue and green are higher after 100 time slots. This is because these are attractors in different attractor sets and continue past states for inactive attractors. In practice, this is not a problem because the attractors uniquely selected are determined by the active attractor set. In the 100 time-slap, the attractor set is switched to one that includes black and green, but in this case, the previously high black attractor is suppressed and the green attractor is selected.

3) Comparison with BAM: Evaluate the percentage of correct responses when the number of attractors is increased in the original BAM and HBAM to see the effect of hierarchy. Here, the number of attractors in the attractor set is fixed at 3, and the overall number of attractors is increased by increasing the number of attractor sets. Here, HBAM performed state update by Parallel Update.

Fig. 3 shows the relationship between the number of attractors and the correct response rate. The horizontal axis is the number of attractors and the vertical axis is the correctness rate. The red line represents the results of HBAM and the blue line represents the results of BAM.



(a) Parallel Update



(b) Serial Update

Fig. 2. Time Series of Decision State in Numerical Simulation

From the figure, the original BAM shows a sharp decline in the correctness rate when the number of attractors increases, while HBAM shows a slower decline in growth rate as the number of attractors increases. as shown in Fig. 1, BAM is optimal as the number of attractors increases. Since HBAM can increase the overall number of attractors while keeping the number of attractors constant, BAM-derived parameters can be used even when the number of attractors increases. This makes HBAM a more scalable model for increasing the number of attractors.

4) Effect of Number of Attractor Set: Given the overall number of attractors, we evaluated the decision-making results while changing the size of the split in order to examine the impact of the number of attractor sets to be split on the decision. Here, a total of 48 attractors were split into 1, 2, 3, 4, 6, 8, 12, 16, 24, and 48 attractor sets. For each partitioning method, we measured the time to convergence of the decision state when the observed values changed and the computation time required for each time slot.

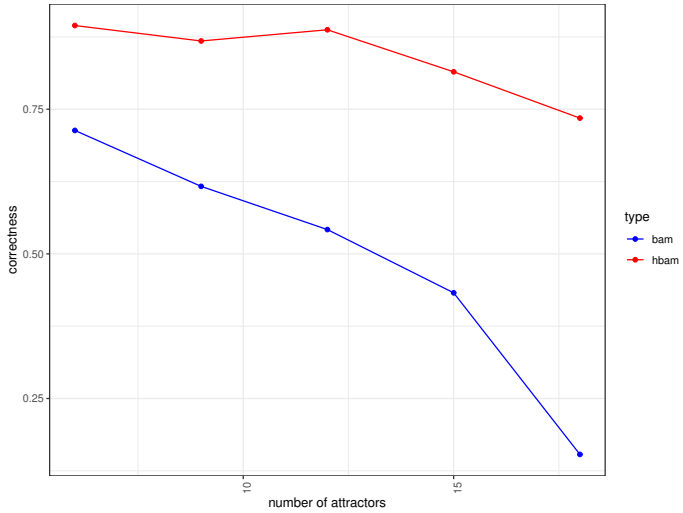
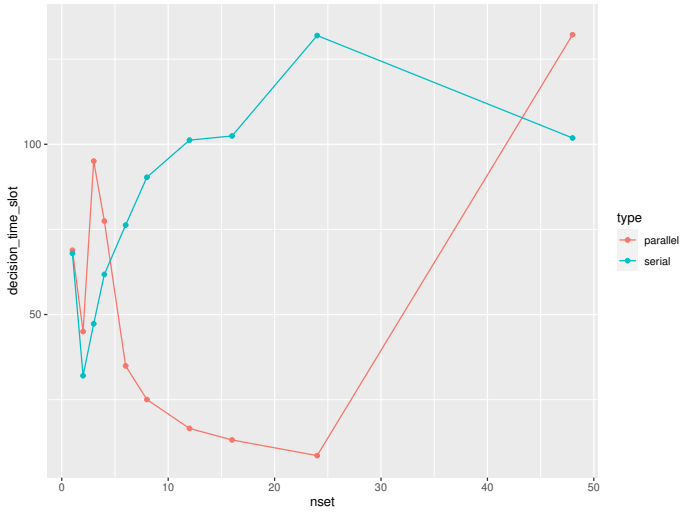
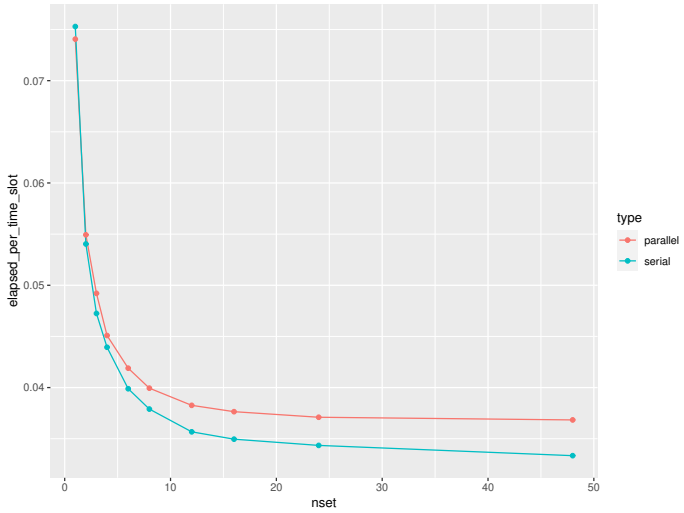


Fig. 3. Comparison between HBAM and BAM with increasing total number of attractors



(a) Required Time Slots to Decide



(b) Calculation Time per Timeslot

Fig. 4. Calculation Time with Parallel/Serial Update and Number of Attractor Sets

Fig. 4 shows the number of timeslots required for decision making and the computation time per timeslot. The horizontal axis is the number of attractor sets, and the vertical axis shows the timeslots and real time (in seconds), respectively. The red line indicates the parallel update case, and the blue line indicates the serial update case.

The figure shows that too few or too many attractor sets increase the time slot required to reach a decision. Appropriate attractor aggressiveness differs between the parallel and serial update cases. In the case of parallel updates, 24 sets are the fastest, and in the case of series updates, 2 sets are the fastest to make a decision.

The computation time per time slot becomes shorter as the number of attractor sets increases. This is because increasing the number of attractor sets reduces the number of attractors per set, and in BAM, the computation time decreases as the number of attractors decreases because the amount of computation depends on the number of attractors. Overall, the computation time for series update is shorter than that for parallel update. This is because only one set of attractors is active and computed in series update, so the computation load is smaller than in parallel update, where the entire attractor set is called on. However, the difference is small, and the effect of the number of attractors in the set on the computation time is clearly larger.

If the number of attractor sets with the shortest decision time is set for each update method, the parallel update is faster than the serial update in terms of both decision speed and computation time per time slot. Based on the above, it is considered that hierarchical updating is suitable while using parallel updating.

VI. Conclusion

In this paper, we propose HBAM to quickly determine the solution to a task assignment problem in an edge-cloud computing environment, which is a mixture of genuine and fluctuating environments, in order to quickly determine the solution in a similar genuine environment in the past. HBAM introduces a hierarchical structure into BAM, which is a model of decision making, and selects similar attractors based on observed values from a small number of attractors in the set while switching multiple attractor sets. This enables the system to maintain a high accuracy rate even when the total number of attractors increases. We also evaluated the state update method and attractor set size associated with the hierarchical structure, and showed that it is better to update attractor sets in parallel and use an intermediate size.

In future research, we will evaluate the performance of HBAM in a simulation environment that mimics an actual edge cloud computing environment. In addition, we will examine what kind of clustering is best for constructing the attractor set.

Acknowledgment

This work was supported by MIC under a grant entitled ”R&D of ICT Priority Technology (JPMI00316)”.

REFERENCES

[1] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[2] S. K. Sharma, I. Woungang, A. Anpalagan, and S. Chatzinotas, "Toward tactile internet in beyond 5g era: recent advances, current issues, and future directions," *Ieee Access*, vol. 8, pp. 56 948–56 991, 2020.

[3] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[4] D. Smilkov, N. Thorat, Y. Assogba, C. Nicholson, N. Kreeger, P. Yu, S. Cai, E. Nielsen, D. Soegel, S. Bileschi *et al.*, "Tensorflow. js: Machine learning for the web and beyond," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 309–321, 2019.

[5] M. Asim, Y. Wang, K. Wang, and P.-Q. Huang, "A review on computational intelligence techniques in cloud and edge computing," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 6, pp. 742–763, 2020.

[6] S. Bitzer, J. Bruineberg, and S. J. Kiebel, "A bayesian attractor model for perceptual decision making," *PLoS computational biology*, vol. 11, no. 8, p. e1004442, 2015.

[7] A. Löffler, A. Sylaidi, Z. Fountas, and P. Haggard, "A hierarchical attractor network model of perceptual versus intentional decision updates," *Nature communications*, vol. 12, no. 1, pp. 1–17, 2021.

[8] L. Deserno, R. Boehme, C. Mathys, T. Katthagen, J. Kaminski, K. E. Stephan, A. Heinz, and F. Schlagenhauf, "Volatility estimates increase choice switching and relate to prefrontal activity in schizophrenia," *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, vol. 5, no. 2, pp. 173–183, 2020.

[9] C. D. Mathys, E. I. Lomakina, J. Daunizeau, S. Iglesias, K. H. Brodersen, K. J. Friston, and K. E. Stephan, "Uncertainty in perception and the hierarchical gaussian filter," *Frontiers in human neuroscience*, vol. 8, p. 825, 2014.

[10] A. I. Jang, R. Sharma, and J. Drugowitsch, "Optimal policy for attention-modulated decisions explains human fixation behavior," *Elife*, vol. 10, p. e63436, 2021.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] T. Parr, G. Pezzulo, and K. J. Friston, *Active inference: the free energy principle in mind, brain, and behavior*. MIT Press, 2022.

[13] S. Bitzer, J. Bruineberg, and S. J. Kiebel, "A Bayesian attractor model for perceptual decision making," *PLOS Computational Biology*, vol. 11, no. 8, p. e1004442, Aug. 2015.

[14] J. Carpenter, P. Clifford, and P. Fearnhead, "Improved particle filter for nonlinear problems," *IEE Proceedings-Radar, Sonar and Navigation*, vol. 146, no. 1, pp. 2–7, 1999.

[15] Y. Chen, Y. Sun, C. Wang, and T. Taleb, "Dynamic task allocation and service migration in edge-cloud iot system based on deep reinforcement learning," *IEEE Internet of Things Journal*, 2022.

[16] J. Lee, H. Ko, J. Kim, and S. Pack, "Data: Dependency-aware task allocation scheme in distributed edge clouds," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7782–7790, 2020.

[17] Q. Xu, Z. Su, M. Dai, and S. Yu, "Apis: Privacy-preserving incentive for sensing task allocation in cloud and edge-cooperation mobile internet of things with sdn," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5892–5905, 2019.