6-2012

iFALCON: A neural architecture for hierarchical planning

Budhitama SUBAGDJA

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Neurocomputing 86 (2012) 124–139

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

iFALCON: A neural architecture for hierarchical planning

Budhitama Subagdja *, Ah-Hwee Tan

School of Computer Engineering, Nanyang Technological University, Singapore

ARTICLE INFO

Article history:
Received 13 June 2011
Received in revised form
25 November 2011
Accepted 17 January 2012
Communicated by N.T. Nguyen
Available online 24 February 2012

Keywords:
Hierarchical planning
Plan learning
Adaptive resonance theory

ABSTRACT

Hierarchical planning is an approach of planning by composing and executing hierarchically arranged predefined plans on the fly to solve some problems. This approach commonly relies on a domain expert providing all semantic and structural knowledge. One challenge is how the system deals with incomplete ill-defined knowledge while the solution can be achieved on the fly. Most symbolic-based hierarchical planners have been devised to allow the knowledge to be described expressively. However, in some cases, it is still difficult to produce the appropriate knowledge due to the complexity of the problem domain especially if the missing knowledge must be acquired online. This paper presents a novel neural-based model of hierarchical planning that can seek and acquire new plans online if the necessary knowledge are lacking. It enables all propositions and descriptions of plans to be computed and learnt simultaneously as inherent features of the model rather than discretely processed like in most symbolic approaches. Using a multi-channel adaptive resonance theory (fusion ART) neural network as the basic building block of the architecture and a new representation technique called gradient encoding, the so-called iFALCON architecture can capture and manipulate sequential and hierarchical relations of plans on the fly. Case studies using blocks world domain and an agent in Unreal Tournament video game demonstrate that the model can be used to execute, plan, and discover new plans through experiences.

1. Introduction

The ability to form and follow plans in an unanticipated condition is an important feature of an autonomous agent. Given a goal condition, a planning agent comes up with a plan which is a sequence of steps that lead from the starting condition to the goal. Computationally, a direct approach to find a plan comprises searching the state space of the problem using a search algorithm usually handled in symbolic manners [1]. However, current models of planning generally assume the availability of partial plans arranged hierarchically as knowledge pre-given by a programmer or a domain expert. This kind of approach is also known as hierarchical planning [2]. The hierarchical planner can be combined with learning from experiences when plans are incomplete [3–7]. This approach may relieve domain experts from crafting complete and correct knowledge for the hierarchical planner.

When the planning and plans acquisition are online, situated, and tightly coupled in a dynamic environment, the process becomes more challenging. Some planning approaches have considered *learning-by-doing* in which the definition or parameters of operators are acquired by observation and practices on the run [8–10]. However, acquiring complex hierarchical structure of plans directly from experiences while the agent is executing or searching for the plans is still a major issue.

Different neural-inspired models or brain-like structures have also been proposed as controllers of the planning processes [11,12]. These approaches still do not capture plans on the run nor flexibly reuse them in different situations. Some other works on neural networks for declarative memory and knowledge have already considered both the biological plausibility and the representational adequacy of reactive planning systems [13] and relating to the state-space search method by demonstrating a simple backward chaining process to search for a plan over the neural network structure [14]. However, the same issue of capturing hierarchical plans on the fly is almost totally untouched.

This paper presents a new neural architecture for hierarchical planning agents featuring learning to capture new plans on the run. The proposed architecture is built to process and handle complex representation like sequences and hierarchical structure which adequately support many high level cognitive functions. The architecture is a composition of simpler multi-channel neural networks as building blocks called fusion ART [15–17]. Unlike most conventional neural network architectures such as multi-layered perceptrons or associative networks, a fusion ART network is continually processing information in cycles of categorizing, matching, learning,

* Corresponding author.
E-mail addresses: budhitama@ntu.edu.sg (B. Subagdja), asahtan@ntu.edu.sg (A.-H. Tan).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2012.01.008

and dynamically allocating new neurons based on some adjustable parameters. The proposed architecture, called iFALCON, combines different fusion ARTs and can map symbolic descriptions into weighted neural connections and uses the knowledge to autonomously find and capture a solution.

The model turns interacting pattern matching processes in the neural network into a state-space search mechanism for planning. As inherent in the neural network, the plans can be processed simultaneously and learnt continuously rather than separately or serially processed like in most symbolic-based approaches. Our contributions to the current state-of-the-art can be described as follows:

- We present a new neural encoding technique to represent sequences and transient hierarchical structure as activation patterns that can be learnt, matched, and read out (recall) in the neural network supporting hierarchical planning processes like plan retrieval, execution, and backtracking.
- We describe how different components of neural networks can be combined to realize a hierarchical planning system that can execute plans and expand subgoals to achieve the overall goal.
- We present an online search mechanism in the neural network to discover alternative plans whenever an impasse condition occurs or the applicable plan for the goal is unknown. This includes the mechanism to capture the successful search episode into a new plan.
- We also demonstrate how planning, learning, and execution can seamlessly interleave with each other in continuous cycles of activity using and forming the dynamic hierarchical structure of plans in the neural network.

The main objective of this paper is neither suggesting a new ultimate planner that can produce optimal solutions nor providing the accurate picture of how a biological brain conveys planning. Instead, the model is developed mainly to explore the solutions for learning and adaptation in plan-based and planning agents using a biologically-plausible mechanisms and structures. Implementations and experiments have been conducted as proof-of-concepts that iFALCON can be set up to solve a planning problem as a hierarchical planner. The implementation of iFALCON demonstrates that the planning can be improved over time while new plans are discovered.

The rest of the paper is organized as follows. Section 2 discusses the process of planning and how plans can be represented and processed. Section 3 discusses how planning can be represented in neural networks. The section describes the self-organizing neural network model called fusion ART used as the building block of the planning architecture. The section also introduces iFALCON as the proposed planning agent architecture. Section 4 shows the results from our implementation of the architecture using the blocks world domain. It confirms that iFALCON can follow plans as prior knowledge while invent and learn new ones when the knowledge is insufficient. Section 5 also demonstrate the use of iFALCON for controlling non-player character (NPC) in a realtime first-person-shooter video game. Section 6 presents the analysis of the performance of iFALCON and explains current limitations of the proposed model and discusses possible extensions to address them. Section 7 summarizes and concludes the work with some discussion on further use and development of the architecture.

2. Hierarchical planning

Planning is the process of finding a course of actions or a *recipe* by taking a goal description, the current state of the environment and available actions to come up with a plan so that if an agent executes this plan in the right order starting from the initial state, the specified goal will be carried out after the execution.

A plan can be defined to consist of preconditions that make the plan applicable, the end results after its execution as the goal conditions to achieve, and the sequence of actions to follow.

Definition 1. A plan π is a tuple $\pi = \langle \mathcal{P}_\pi, \mathcal{G}_\pi, \mathcal{A}_\pi \rangle$ in which $\mathcal{P}_\pi \in 2^{\mathcal{S}}$, $\mathcal{G}_\pi \in 2^{\mathcal{S}}$, and $\mathcal{A}_\pi \in 2^{\Delta}$. A proposition $p \in \mathcal{S}$ and a step of action $a_i \in \Delta$.

Given a finite set of plan (plan repository) Π, $\pi \in \Pi$ can be selected for execution if it is applicable at the current moment and the end results fulfils the goal conditions to achieve.

Definition 2. Given $P \in 2^{\mathcal{S}}$ the current state of affair and $G \in 2^{\mathcal{S}}$ the goal conditions to achieve, a plan $\pi \in \Pi$ can be selected for execution if $\mathcal{P}\pi \subseteq P$ and $\mathcal{G}_\pi \supseteq G$.

A classical approach of planning can be applied to find a new plan if no applicable plan can be found directly in Π. With goal G and initial state P, a *forward-chaining* search can be conducted by firstly selecting an applicable plan π_t ($\mathcal{P}_{\pi_t}$) and appending the actions $\mathcal{A}_{\pi_{t+1}}$ of another plan π_{t+1} such that $\mathcal{P}_{\pi_{t+1}} \subseteq \mathcal{G}_{\pi_t}$ iteratively until a plan π_i is appended in which $\mathcal{P}_{\pi_i} \subseteq \mathcal{G}_{\pi_{i-1}}$. On the other hand, a *backward-chaining* approach can also be applied by firstly selecting π_i wherein $\mathcal{G}_{\pi_i} \supseteq G$ and iteratively $\mathcal{A}_{\pi_{i-1}}$ is inserted at the beginning of $\mathcal{A}_{\pi_i}$ wherein $\mathcal{G}_{\pi_{i-1}} \supseteq \mathcal{P}_{\pi_i}$ until π_t that $\mathcal{G}_{\pi_t} \supseteq \mathcal{P}_{\pi_{t+1}}$ and $\mathcal{P}_{\pi_t} \subseteq P$.

The basic forward or backward search method may be impractical for agents situated in a complex dynamic environment as the number of possible branches of alternatives in any state may be intractable. Different techniques like heuristic-guided search [19,20], and hierarchical planning [2] are usually employed to deal with the complexity.

In hierarchical planning [2] (it is often known also as Hierarchical Transition Network), plans are selected and composed hierarchically from a memory storage or repository by employing *non-primitive* actions to carry out subplans during execution. The approach can take less computational efforts as it does not have to search all possible sequences but focus only options as prescribed by subplans [21,22]. The issue is no longer to find the right sequence to achieve the goal, but to choose the right goal (deliberation) and the right plan (means-end reasoning) at the right moment as parts of the execution.

The non-primitive action usually corresponds to an instruction to achieve a subgoal in which a new goal is posted so that the execution process will start the deliberation to reconsider the goal and if necessary starting the means-end reasoning to select a plan to achieve it.

Definition 3. A non-primitive subgoal action $\overrightarrow{a_i}^{G'} \in \Delta$ will expand the current step of action into actions $\mathcal{A}_{\pi_{sub}}$ in which $\mathcal{P}_{\pi_{sub}} \subseteq P$ and $\mathcal{G}_{\pi_{sub}} \supseteq G'$. P and G' are the current state of affair and the new subgoal as described by the non-primitive $\overrightarrow{a_i}^{G'}$ respectively.

The application of subgoal actions in a plan may also produces a transient structure of selected plans and goals pending achievements. In some agent architectures (e.g. PRS [23], BDI agent [24]) this structure is called *intention structure*. The hierarchical structure of plans allows the process of finding the solution to achieve the goal to interleave with the action executions. The expansion of a subgoal can be directly followed by a primitive action that directly changes the environment.

The hierarchical approach requires an applicable (partial) plan to be available or known so that each subgoal can be expanded. Otherwise, the agent cannot do anything unless an external

planning process is applied or a learning mechanism is conducted to add the collection of partial plans at runtime. One of the issue to be tackled in this paper is to combine the hierarchical planning approach with the classical search approach whenever an impasse condition occurs in which no plan for a certain goal can be found. The paper also includes the solution on how a new discovered sequence of actions from the search process can be retained so that it can be reused at a later time. We suggest a neural network architecture that can solve these problems in hierarchical planning.

3. Neural network planning

In a symbolic form, a state or a condition can be expressed as a conjunction of logical propositions. The set of propositions expressing the state can also be represented as a vector suitable for neural networks.

Definition 4. An input (state) vector $\mathbf{I}=(I_1,I_2,\ldots,I_n)$ represents a state $s\in\mathcal{S}$ wherein I_i is a real value and $0\le I_i\le 1$. A proposition p can be associated with a subset of $\mathbf{I}$ or $p\subseteq \mathrm{I}$.

A vector $\mathbf{I}$ can be used as an input (or output) to the neural network. As a state description (e.g. perception), $\mathbf{I}$ can be fully specified in which each proposition is assigned with a particular value (e.g. 1 or 0 to reflect binary logic, or a real value if fuzzy logic is applied instead). However, in a symbolic approach, not all propositions may be specified or some values may be unknown but can be omitted from the specification.

The state vector $\mathbf{I}$ above can also be made to compromise unspecified values by applying *complement coding* [25].

Definition 5. *Complement coding* is applied to the input vector $\mathbf{I}$ by augmenting each element I_i with its complement $\bar{I}_i=1-I_i$ such that a proposition p_i corresponds to a pair $(I_i,\bar{I}_i)$. Both I_i and $\bar{I}_i$ are elements of $\mathbf{I}$.

This encoding approach offers more expressive power to represent a proposition and supports generalization. One direct way to generalize a proposition is by setting an equal pair of the complemented values at their extreme. For example, a pair (0,0) might mean *do not-know* value condition and (1,1) would mean *do not-care* condition. It is also possible to express a range of values beyond a specific boolean or analog value. When $I_i\neq 1-\bar{I}_i$, the corresponding p_i becomes less specified so that $(I_i,\bar{I}_i)\equiv I_i\ge p_i\ge 1-\bar{I}_i$. This feature of generalization will be explained further later in this paper.

3.1. Fusion ART: the building blocks

The proposed model in this paper is based on fusion ART [16] neural network which can be viewed as a derivation of adaptive resonance theory (ART) neural network [26] extended with multiple input fields or channels. The mechanisms and basic principle of ART network have been suggested to be pervasive in many areas in the brain [27]. The mechanism is based on the view that learning comprises classification and expectation apart from the knowledge update. As shown in Fig. 1(i) the basic ART network consists of two neural fields: input field and neural field connected with bi-directional weighted connections.

Two complementary processes are employed: bottom-up classification and top-down matching. The bottom-up process categorizes the input pattern through a competitive activation by employing a winner-take-all strategy. The top-down matching, then, expects and judges the familiarity of the input pattern based on the degree to which it fits with the current selected category. The two processes can be said to resonate with each other if a matching category is found so that one process reinforces and is reinforced by the other. If the input pattern does not meet the matching criteria of the top-down process, the current category is reset and the bottom-up categorization continues until a matching category is found. Unsupervised learning is employed when a *resonance* occurs by updating the connection weights between the input vector and the matching category. However, as one important feature of ART, a new category node is allocated if no matching category can be found so that the network can grow dynamically for every novel patterns encountered.

In this way, the learning in ART network can be fast but stable by avoiding catastrophic interference between prior learnt knowledge and novel patterns. As another useful feature of ART, the level of generalization and discrimination against novel patterns is controllable by adjusting the vigilance level of the matching criteria either offline or dynamically at runtime. Learning, classification, and matching are not conducted separately in different phases. Instead, the pattern matching and learning are integrated parts of the execution cycles in the overall network activity.

The proposed planning model makes use of several multiple channel ARTs (fusion ARTs) as building blocks. As shown in Fig. 1(ii), Fusion ART extends the original ART model by employing multiple input fields or channels [16]. By employing multiple input (output) channels, the ART network can also be enhanced to comprise different learning schemes like supervised and reinforcement learning in addition to the basic unsupervised learning. Various types of multi-channel ART has been used to realize rule-guided agents that employ online reinforcement learning [15–17]. For example, FALCON [15] employs reinforcement learning by configuring the input fields to act as input *state*, output *action*, and *reward* feedback so that it learns actions policy based on external evaluative feedback.

Similar to its original predecessor, fusion ART network employs the iterative resonance search process to find a matching category. However, each input field applies independent parameters which enables it to process different input modalities. Depending on the task domain, a fusion ART network may also apply different types of vector encoding on its different input fields. Some fields may apply *complement coding* to prevent *category proliferation* and enable generalization [25], but the others may not.

In what follows, how fusion ART categorizes, predicts, and learn patterns will be formulated and explained in more detail.

Definition 6. F_1^k and F_2 are the kth input (output) field and the category field of fusion ART (Fig. 1) respectively for $k=1,\ldots,n$. Let $\mathbf{x}^k=(x_1^k,x_2^k,\ldots,x_n^k)$ be the activity vector of F_1^k receiving the input vector $\mathbf{I}^k$ and $\mathbf{y}=(y_1,y_2,\ldots,y_n)$ is the activity vector of F_2. Let $\mathbf{w}_j^k$

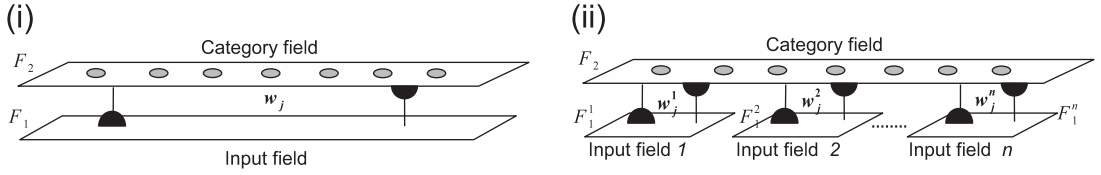

Fig. 1. (i) ART neural network and (ii) fusion ART architecture.

be the weighted connection vector associating the jth node in F_2 (jth element of $\mathbf{y}$) to elements of $\mathbf{x}^k$ in F_1^k.

The dynamic of fusion ART depends on several parameters, each is assigned to different input (output) fields.

Definition 7. F_1^k is associated with choice parameter $\alpha^k > 0$, learning rate $\beta^k \in [0,1]$, contribution parameter $\gamma^k \in [0,1]$, and vigilance parameter $\rho^k \in [0,1]$.

For each channel k, $\alpha^k > 0$ and $\gamma^k \in [0,1]$ regulate the node activation level in F_2 during the bottom-up activation after the presentation of vector $\mathbf{x}^k$ in F_1^k. The vigilance parameter $\rho^k \in [0,1]$ sets the level of tolerance for value differences during the top-down matching process. The Learning rate parameter $\beta^k \in [0,1]$ sets the rate of the weighted connections change during learning.

The dynamics of fusion ART can be considered as continual cycles consisting of some basic operations

Definition 8. *Choice function* T_j returns the activation value of category j such that

$$T_j = \sum_{k=1}^{n} \gamma^k \frac{|\mathbf{x}^k \wedge \mathbf{w}_j^k|}{\alpha^k + |\mathbf{w}_j^k|} \tag{1}$$

where the fuzzy AND operation $\wedge$ is defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv \min(p_i, q_i)$, and the norm $|\cdot|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$.

Definition 9. *Template matching* m_j^k is the matching value or similarity of category j with the input $\mathbf{x}^k$ such that

$$m_j^k = \frac{|\mathbf{x}^k \wedge \mathbf{w}_j^k|}{|\mathbf{x}^k|} \tag{2}$$

Given the input vector, the ART network search and select a category J of F_2 field that fits with *resonance condition.*

Definition 10. A node J of F_2 field is in *resonance condition* if and only if

$$T_J = \max\{T_j : \forall k, m_j^k \geq \rho^k, \text{ for all } F_2 \text{ category } j\} \tag{3}$$

Given the selected node J, learning takes place by modifying the weight vector $\mathbf{w}_J^k$.

Definition 11. *Template learning* modifies the weights associated with category J such that

$$\mathbf{w}_J^{k(\text{new})} = (1-\beta^k)\mathbf{w}_J^{k(\text{old})} + \beta^k(\mathbf{x}^k \wedge \mathbf{w}_J^{k(\text{old})}) \tag{4}$$

The corresponding weight vector of the chosen F_2 node J can be readout into the input field F_1^k such that $\mathbf{x}^{k(\text{new})} = \mathbf{w}_J^k$. Depending on the domain problem, the readout may also be the fuzzy AND of their original values and their corresponding weight vectors such that $\mathbf{x}^{k(\text{new})} = \mathbf{x}^{k(\text{old})} \wedge \mathbf{w}_J^k$.

If no existing F_2 category can be found in resonance condition with the current input, a new category is recruited to represent the current input pattern. This implies that the ART network can grow to accommodate the incoming stream of different input patterns. The growth rate of the categories depends on how much the incoming patterns differ from one another and is adjustable through adjusting the vigilance parameters (ρ^k). Lower vigilance may tolerate differences more than the higher one and hence lead to a slower growth.

Fusion ART can also support generalization of activation values by applying *complement coding* as described above. The generalization can be achieved by adjusting the vigilance parameter ρ^k so that slightly different input patterns will still activate the same category. As in *complement coding*, the value of a stored attribute or proposition can be paired so that the ART can generalize the value as a range of values.

Definition 12. Let $\mathbf{I}^k$ be the input vector for $F_1^k, I_i^k \in [0,1]$. $\mathbf{I}^k$ is augmented with $\overline{\mathbf{I}}^k$ such that $\overline{I}_i^k = 1 - I_i^k$. The activity vector $\mathbf{x}^k$ of F_1^k thus augments the input vector $\mathbf{I}^k$ with its complement $\overline{\mathbf{I}}^k$ which are learnt as a $\mathbf{w}_j^k$. Let $(w_{ij}^k, \overline{w}_{ij}^k)$ be the corresponding pair of $\mathbf{w}_j^k$. The value of the connection becomes less specified when $w_{ij}^k \neq 1 - \overline{w}_{ij}^k$.

Lemma 1. *For the pair* $(w_{ij}^k, \overline{w}_{ij}^k)$ *of* $\mathbf{w}_j^k$ *described above, if* $w_{ij}^k \neq 1 - \overline{w}_{ij}^k$, *any corresponding complemented input pair* $(I_{ij}^k, \overline{I}_{ij}^k)$ *will have a maximum matching value or always in resonance as long as* $w_{ij}^k \geq I_{ij}^k \geq 1 - \overline{w}_{ij}^k$.

Proof. Let the pair $(x_{ij}^k, \overline{x}_{ij}^k) \equiv (I_{ij}^k, \overline{I}_{ij}^k)$. It is clear that if $x_{ij}^k \leq 1 - \overline{w}_{ij}^k$ then $\overline{x}_{ij}^k \leq \overline{w}_{ij}^k$. Thus, $(x_{ij}^k, \overline{x}_{ij}^k) \wedge (w_{ij}^k, \overline{w}_{ij}^k) = (x_{ij}^k, \overline{x}_{ij}^k)$ such that *template matching* $m_j^k = |\mathbf{x}^k \wedge \mathbf{w}_j^k| / |\mathbf{x}^k| = |\mathbf{x}^k| / |\mathbf{x}^k| = 1$. □

3.2. Representing sequences and hierarchical relationships

The standard configuration of fusion ART can only learn and categorize patterns that occur simultaneously or assumed to be occurred at the same time and no information about time or the order of the incoming patterns. A suitable way to capture temporal patterns has actually been suggested in some early works of the ART neural network model. Grossberg has proposed *item and order working memory* model in which the temporal order is encoded in the *relative activity* of different node populations [28]. The idea is that successive item categories that are activated through time can be sequentially stored in working memory as a temporally evolving spatial pattern of activity across working memory cells. At a later time, the sequence can be reproduced by rehearsing the largest activities earliest and suppressing the item that has just been rehearsed one at a time. The activation value of each category multiplicatively modifies the activity of all previous items. The item and order principle has also been used in STORE (Sustained Temporal Order Recurrent) network to mimic the behavior of working memory [29] and recently in LIST PARSE suggesting its pervasiveness in various brain areas [27]. As a part of the proposed model, the fusion ART is extended to associate and group pattern occurring across time. A similar technique with the item and order model can also be applied to fusion ART for dealing with sequential and hierarchical patterns. However, instead of multiplications, the proposed model employs a temporal encoding technique called *gradient encoding* to retain and process temporal order of neural activations which simplifies the original *item and order* using simple constant increments (decrements) to determine the activation values.

Definition 13. In *gradient encoding*, the value of the selected category J in a category field is set to τ such that $y_J = \tau$, and τ is updated so that $\tau^{(new)} = \tau^{(old)} + \upsilon$ or $\tau^{(new)} = \tau^{(old)} - \upsilon$, where $0 \leq \tau \leq 1$, $0 < \upsilon < 1$. Given that $y_{j_t} = \tau_t$ wherein t is the relative time point and j_t is the category selected at t, if $\tau_0 \geq 1$ and $\tau_{t+1} = \tau_t - \upsilon$ or $\tau_{t+1} = 0$ whenever $\tau_t - \upsilon \leq 0$, then $\mathbf{y}$ reflects the relative order of category selection (*primacy gradient*) such that $y_{j_t} > y_{j_{t+1}} > y_{j_{t+2}} > \cdots > y_{j_{t+m}}$. If $\tau_0 \leq 0$ and $\tau_{t+1} = \tau_t + \upsilon$ or $\tau_{t+1} = 1$ whenever $\tau_t + \upsilon \geq 1$, then $\mathbf{y}$ reflects the relative reverse order of category selection (*recency gradient*) such that $y_{j_t} > y_{j_{t-1}} > y_{j_{t-2}} > \cdots > y_{j_{t-n}}$.

Fig. 2 shows different types of sequential ordering in gradient encoding compared with the standard winner-take-all activation.

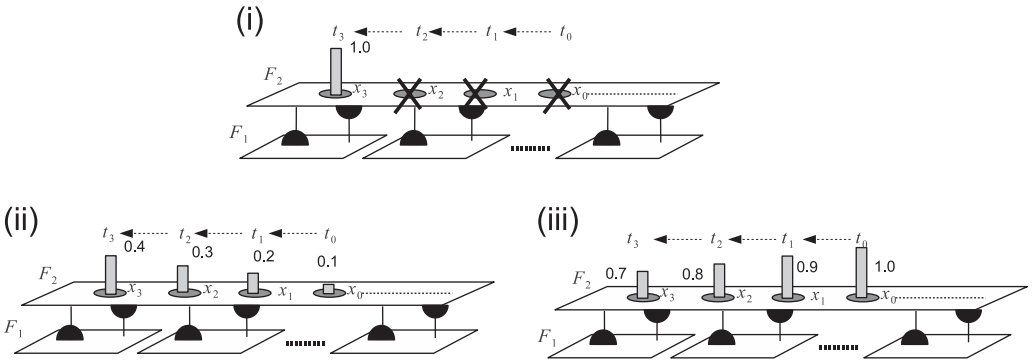

Fig. 2. (i) Winner-take-all activation; (ii) recency gradient activation pattern, and (iii) primacy gradient activation pattern.

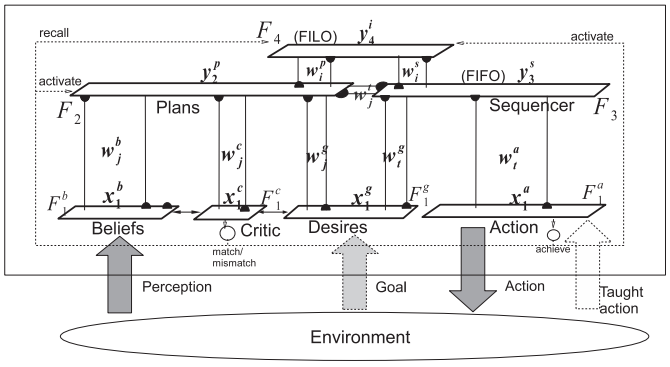

Fig. 3. iFALCON architecture.

The analog pattern formed following the gradient encoding can directly be learnt and grouped as a category using the same mechanism of resonance search cycle in fusion ART. The sequential pattern can be considered as the input to a fusion ART network. In the proposed planning architecture described next, some connected layers of fusion ARTs employ gradient encoding in their category field allowing another fusion ART to learn the sequential patterns.

3.3. iFALCON: a plan-based neural network

iFALCON is a neural architecture that arranges and groups different fusion ART networks to emulate the process of planning and plan execution. As shown in Fig. 3, iFALCON consists of four input (output) fields and three category fields. F_1^b, F_1^g, F_1^c, and F_1^a denote *beliefs*, *desires*, *critic*, and *action* fields respectively. The *beliefs* field corresponds to the state of the environment and is continuously updated by the incoming perception. The *desires* field represent the goal to be achieved and can be set up externally at the beginning of the planning to achieve the domain objective. The *critic* reflects the difference between the values in *beliefs* and *desires*. The *critic* value may activate the resonance search in F_2 or F_4 to select a plan or restore the last condition pending achievements respectively. The *action* field represents the action to take at a moment to update the environment. As parts of the hierarchical planning system, the activation of *achieve* node corresponding to an abstract action in F_1^a activates the resonance search in F_4 and F_2 to initiate the subgoal expansion.

As shown in Fig. 4, iFALCON can be seen as a composition of three different interconnected fusion ART modules. The structure and characteristics of each module can be explain as follows:

- **Plan repository** is a fusion ART module to store and retrieve plans. A node j in the category field F_2 corresponds to a plan description π_j and is connected to nodes in F_1^b, F_1^g, F_1^c, and F_3 via connections $\mathbf{w}_j^b$ (the precondition $\mathcal{P}_{\pi_j}$), $\mathbf{w}_j^g$ (the postcondition $\mathcal{G}_{\pi_j}$), $\mathbf{w}_j^c$ (the critic value), and $\mathbf{w}_j^t$ (the body $\mathcal{A}_{\pi_j}$) respectively as the main attributes of a plan. Selecting an applicable plan to achieve the goal corresponds to the resonance search to select a node j in F_2 in a winner-take-all manner given the goal and the current situation as vector $\mathbf{x}_1^g$ and $\mathbf{x}_1^b$. A new plan will be stored or inserted automatically if the search can not find any existing match.
- **Actions controller** is a fusion ART module to transiently store and replay actions according to their order of presentations. A node t in F_3 is connected to nodes in F_1^g and F_1^a via connections $\mathbf{w}_t^g$ (the subgoal for the abstract action) and $\mathbf{w}_t^a$ (the action) respectively. Nodes activations in F_3 ($\mathbf{y}_3^s$) follow the primacy gradient as described previously.
- **Working memory** is a fusion ART module to transiently store and reproduce the status of the planning process. A node i in F_4 is connected to nodes in F_2 and F_3 via connections $\mathbf{w}_i^p$ (the selected plan) and $\mathbf{w}_i^s$ (the state of the actions sequence) respectively. Nodes activations in F_4 ($\mathbf{y}_4^i$) follow the recency gradient as described previously. Each selection is given an activation value in an increasing order. To restore the status of the plan and the sequence, the maximum node in F_4 is selected and readout.

As a fusion ART network, each module is attached with parameters that usually assigned to input (output) fields (α^k, γ^k, ρ^k, and β^k in which k refers to the label of the corresponding the input or output field). The parameters are also assigned with the index of the corresponding category field such that α_f^k, γ_f^k, ρ_f^k, and β_f^k where f corresponds the label of the category field. For example, parameters for the goal field in the plan repository module (F_2 category field) are α_2^g, γ_2^g, ρ_2^g, and β_2^g.

3.4. Mapping hierarchical plans to neural networks

A plan can be directly encoded to the neural network of iFALCON by mapping propositions, action symbols, and the action sequence in the plan attributes into their corresponding neural connections in the plan repository and action controller modules. It is also possible to apply a supervised learning method by presenting possible inputs, goals, actions, and outcomes as learning samples in the appropriate order. Given plan π, several steps can be pursued to encode π into the neural network as follows:

1. The patterns representing $\mathcal{P}_\pi$ and $\mathcal{G}_\pi$ are presented to F_1^b and F_1^g as vectors $\mathbf{x}_1^b$ and $\mathbf{x}_1^g$ respectively in which a node j is then selected and activated in F_2 based on the resonance search.
2. For all actions in $\mathcal{A}_\pi$, the pattern representation of each action is subsequently presented to F_1^a as vector $\mathbf{x}_1^a$ and F_1^g as vector $\mathbf{x}_1^g$

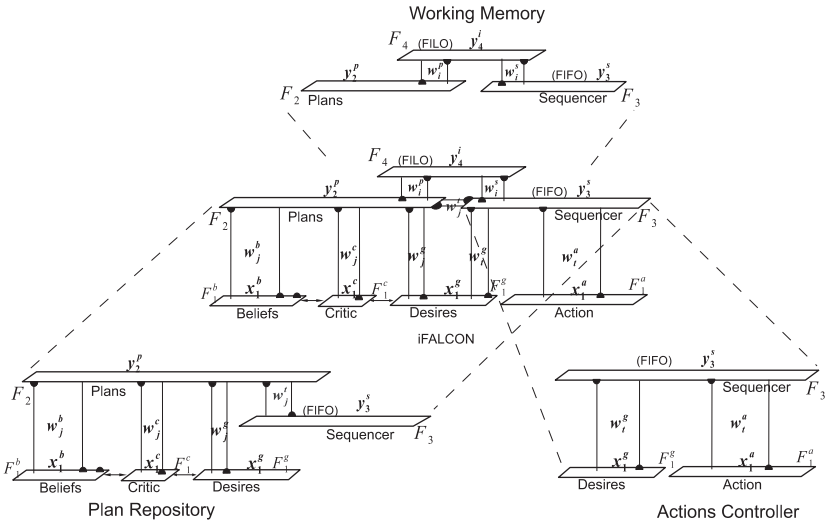

Fig. 4. iFALCON consists of a number of interconnected fusion ART modules.

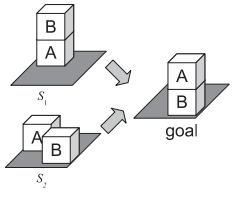

Fig. 5. Simple blocks world.

for a subgoal (achieve) action while activating a node t in F_3 following the primacy gradient.

3. Based on the selected plan node j in F_2, the sequential pattern of actions in F_3 is associated with the plan so that $\mathbf{w}_j^t = \mathbf{y}_3^s$.

As an illustration, consider a simplified blocks world domain like in Fig. 5 in which the target is to stack blocks from one configuration (a state s_1 or s_2) to a different configuration (the goal). The domain can be made to use the following propositions: `A_On_B`, `B_On_A`, `Clear_A`, `Clear_B`, `A_On_Table`, and `B_On_Table` which means that block A is on block B, block B is on block A, block A is clear, block B is clear, block A is on the table, and block B is on the table respectively. The domain also use symbols of actions as follows: `Putdown_A`, `Putdown_B`, `Stack_A_On_B`, `Stack_B_on_A`, and `Achieve`. The symbols for actions are assumed to be self-explained (e.g. `Putdown_A` means put block A down to the table and `Stack_A_On_B` refers to stacking block A on block B). The `Achieve` action is the subgoal action.

For example, given a symbolic description of a simple plan or an operator (consisting of a single action) as follows:

```
{goal: [B_On_Table],
pre: [-B_On_Table, Clear_B],
body: [Putdown_B]}
```

The `goal`, `pre`, `body` are the goal, preconditions, and the sequence of actions respectively. The minus ('−') sign in the description means negation of the proposition. This plan description can be mapped into its corresponding neural connections in iFALCON as illustrated in Fig. 6 following the encoding steps describe above. A connection with a solid line represents 1 (a positive proposition) in the complement coding scheme. A dashed line represents 0 or a negative proposition. Other connections are not shown which correspond to *do-not-care* condition for those employing complement coding or to a zero value for normal encoding. The critic and its encoding in F_1^c is omitted for simplification.

A plan may consist of a sequence of actions rather than just a single step. For example, given a symbolic description of a more generic plan to solve the above blocks world problem as follows:

```
{goal: [A_On_B],
pre: [],
body: [{achieve:[On_A_Table]}, Stack_A_On_B]}
```

Applying the encoding steps above, the plan description can be mapped into its corresponding neural connections as illustrated in Fig. 7. It shows that the sequence is expressed as graded weights following the gradient encoding scheme. For a subgoal action, the connections comprise the *desires* field (F_1^g) besides *action* (F_1^a).

3.5. Plan selection and execution

When all plans are encoded properly as neural connections, it is possible to start the deliberation and execution cycles. The main operations for the basic hierarchical planning processes in iFALCON can be described as follows:

- *Goal monitoring* is the comparison between input vector $\mathbf{x}_1^b$ as the current situation P and $\mathbf{x}_1^g$ as the goal to achieve G. The comparison is conducted by applying $m_g = |\mathbf{x}_1^g \wedge \mathbf{x}_1^b| / |\mathbf{x}_1^g| \geq \rho^e$ wherein ρ^e is the vigilance parameter for goal evaluation. The planning continues only if $m_g < \rho^e$. Otherwise, it finishes or backtracks to its parent (super plan) in the hierarchy. The critic vector $\mathbf{x}_1^c$ is updated with the value m_g.
- *Plan selection* is selecting a plan as a category in F_2 (plan repository) through a resonance search based on $\mathbf{x}_1^b$ (the current situation P), $\mathbf{x}_1^g$ (the goal G), and $\mathbf{x}_1^c$ as the critic. $\mathbf{y}_3^s$ is then updated with the action sequence readout from the selected plan in F_2.
- *Plan execution* performs the steps in the plan body through cycles of reading out and resetting the maximum activations in $\mathbf{y}_3^s$ given the selected plan in F_2 while the action in $\mathbf{x}_1^a$ is carried out.
- *Subgoaling* temporarily stores the current plan ($\mathbf{y}_2^p$) and actions ($\mathbf{y}_3^s$) by a resonance search to select a category in F_4 following the recency gradient and replaces the current goal ($\mathbf{x}_1^g$) with

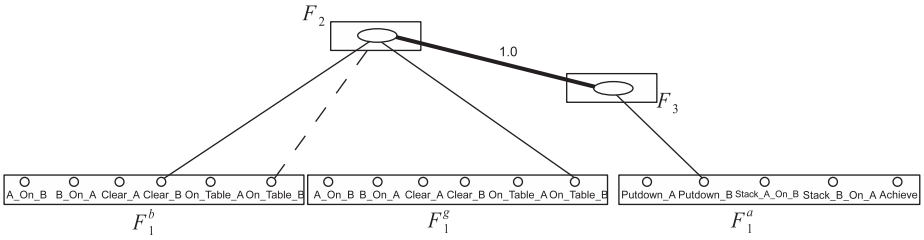

Fig. 6. Mapping a symbolic plan to the neural network.

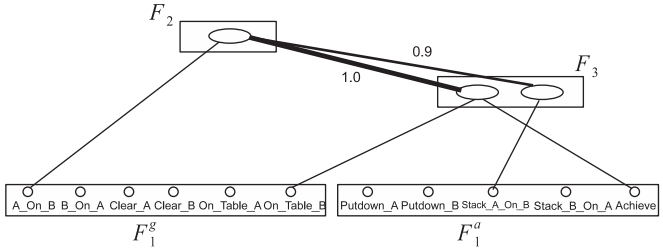

Fig. 7. Mapping a symbolic plan consisting multiple steps to neural networks.

the subgoal readout from the current action. A successful subgoal will be followed by *backtracking* in which $\mathbf{y}_2^p$, $\mathbf{y}_3^s$, and $\mathbf{x}_1^g$ are subsequently readout from the maximum category in F_4 and F_2.

Based on Definitions 8–10, it implies that a plan is selected through resonance search based on the similarity of its preconditions, critic, and postconditions to the current situation, goal, and the expected critic value. To select a plan, the criteria for the critic value can be set to maximum or 1 which also means that the plan selected will have the maximum match value between the perceived situation and the goal. γ_1^k and α_1^k parameters place different weights to the importance of different input fields. A larger γ_1^k can mean a more important input field as a cue or criteria to select a plan. α_1^k, on the other hand, regulate the activation values to ensure that each element is normalized or $0 \le y_{2j}^p \le 1$.

The plan to be selected is the one with the highest choice function that matches with the current situation and goal. The template matching described above implies that a plan π_j with the highest activation value will be selected if $\mathcal{P}_{\pi_j} \subseteq P$ and $\mathcal{G}_{\pi_j} \subseteq G$.

Lemma 2. *Given that each proposition in the current situation P, the goal G, the plan precondition* $\mathcal{P}_{\pi_j}$*, and postcondition* $\mathcal{G}_{\pi_j}$ *correspond to an element (or a complementary pair) in* $\mathbf{x}_1^b$, $\mathbf{x}_1^g$, $\mathbf{w}_{1j}^b$, *and* $\mathbf{w}_{1j}^g$ *respectively, the plan* π_j *will always have the highest match value as long as* $\mathcal{P}_{\pi_j} \subseteq P$ *and* $\mathcal{G}_{\pi_j} \subseteq G$.

Proof. $\mathcal{P}_{\pi_j} \subseteq P$ implies that $\mathbf{x}_1^b \wedge \mathbf{w}_{1j}^b = \mathbf{x}_1^b$ and so $\mathcal{G}_{\pi_j} \subseteq G$ implies that $\mathbf{x}_1^g \wedge \mathbf{w}_{1j}^g = \mathbf{x}_1^g$. If $\mathcal{P}_{\pi_j} \subseteq P$, $m_j^b = \mathbf{x}_1^b / \mathbf{x}_1^b = 1$ or the match value is maximum. The same thing holds between the postcondition and the goal. □

It also follows that based on $m_g = |\mathbf{x}_1^g \wedge \mathbf{x}_1^b| / |\mathbf{x}_1^g| \ge \rho^e$, goal G is always achieved whenever $P \supseteq G$.

To illustrate the mechanism of iFALCON in executing plans, Fig. 8 shows the trace of execution during a plan execution episode for the blocks world domain. Assuming the plans like illustrated in Figs. 6 and 7 have been encoded in the network. Five execution cycles are shown to solve the blocks world problem as shown in Fig. 5. The trace includes the process of *subgoaling* and *backtracking*. At the beginning (Fig. 8(i)), the goals and the current situation are compared based on the match function above (step 1). If the match value is still below vigilance ρ^e, the plan selection process starts and the sequence of actions of the selected plan are readout to the sequencer field (F_3) for execution (steps 2–3). The plan execution process follows by reading out and resetting the maximum category node in F_3 to the corresponding input fields while the readout pattern in the action field is carried out (steps 3–4). However, as the first step of actions is an achieve (subgoal) action, the plan and the sequence are stored in working memory and the subgoal overrides or replaces the current goal afterwhile the plan repository and the sequencer field are reset (steps 5–7).

Based on the new subgoal, another goal monitoring, plan selection, and execution continue until the main goal is achieved (Fig. 8(ii)–(v)). When the achieved goal is a subgoal, *backtracking* is conducted by reading out the maximum category node in F_4 to restore the parental plan and action sequence in the hierarchy as $\mathbf{y}_2^p$ and $\mathbf{y}_3^s$ respectively (Fig. 8(iii)). The example shows that iFALCON is functional as a hierarchical planning system as long as plans to solve the problem domain are sufficiently provided.

3.6. Online planning and learning

The basic execution cycle described above still assume that the available knowledge (plans) are complete without the chance that the agent make mistakes. The process can not deal with the situation when no plan can be found to solve the current goals. To handle the lack of knowledge or insufficient knowledge, there are several features that should be added to the basic hierarchical planning model as follows:

- *Failure detection*. This feature is to identify if no plan can be found to achieve the current goal or the selected plan fails to achieve the goal. In iFALCON this detection function can be achieved by checking whether F_3 field has any activation or not after the resonance search is conducted to select a plan in F_2. A failure of the plan execution can also be detected if $m_g < \rho^e$ but there is no activation in F_3 ($\mathbf{y}_3^s = \mathbf{0}$). It should be noted that a category node F_2 is always selected either as an existing plan category or as a new allocated one if no existing plan can be found.
- *Hypothesis generation*. When a plan failure has been detected, another plan can be hypothesized as a way around or an intermediary solution. This can be done by compromising the criteria to select a plan π' and taking $\mathcal{G}_{\pi'}$ as the hypothetical subgoal to be appended or inserted as a new action step. The sequence of hypothetical actions can be stored temporarily to be tested further as a part of the process of searching for a novel solution. One way to compromise the criteria used in our model is by lowering ρ_1^g so that another applicable plan that may not fully achieve the current goal can still be selected.
- *Plan capture*. If the hypothetical actions eventually achieve the goal, a new plan for those actions can be learnt or stored in the

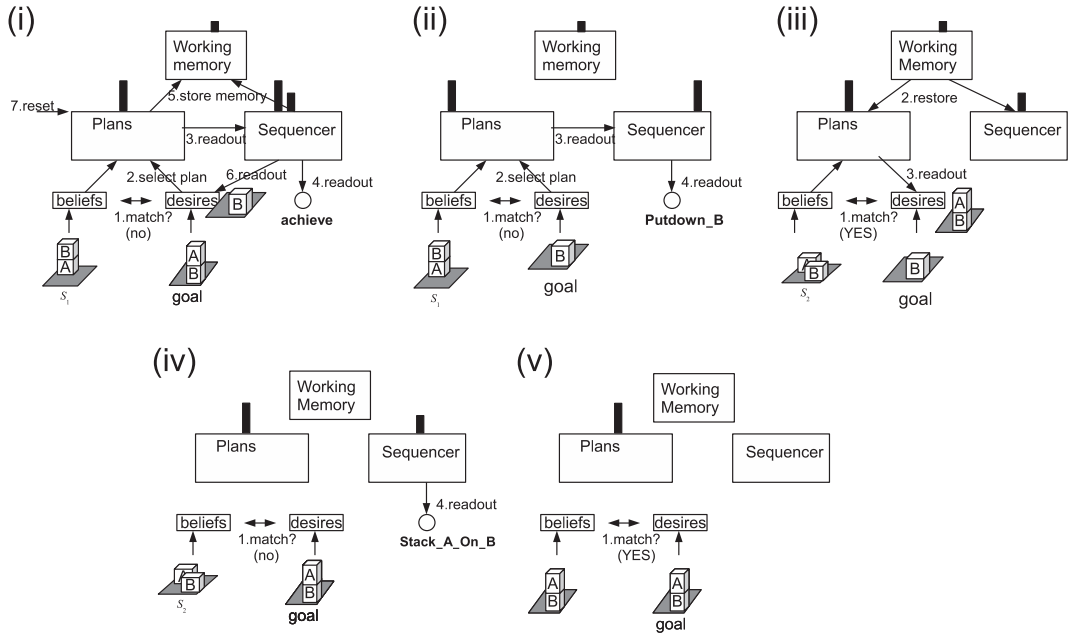

Fig. 8. The execution trace of solving a simple blocks world problem based on the basic hierarchical planning algorithm of iFALCON. Five cycles of execution are required to achieve the goal from the initial state s_1.

plan repository. The hypothetical actions as the results of the search process can be identified by checking whether a successful achievement of a plan still have a maximum activation value in F_3 field or not. The maximum activation in the sequence after a successful attempt to achieve the goal means that a sequence of actions has just been transiently stored as the hypothesis.

Every time no plan can be found from the resonance search, a new empty plan is allocated automatically as an inherent feature of fusion ART to solve the given problem goal. Further search processes can be conducted by firstly storing the new plan and find an intermediate solution with a reduced vigilance factor for the goal. By gradually reducing the goal vigilance ρ_1^g, a plan π_j' may be selected with a relaxed criteria such that $\mathcal{P}_{\pi_j} \subseteq P$ and $\mathcal{G}_{\pi_j} \cap G$. It can also be said that $|\mathcal{G}_{\pi_j} \cap G| \equiv |\mathbf{x}_1^g \wedge \mathbf{w}_{1j}^g| / |\mathbf{x}_1^g|$.

Lemma 3. *Given a plan π_j and let $|\mathcal{G}_{\pi_j} \cap G|^{\pi_j}$ be the number of intersecting propositions between postcondition $\mathcal{G}_{\pi_j}$ and the current goal G so that π_j can be selected, the goal vigilance ρ_1^g is proportional to the number of propositions or $\rho_1^g \propto |\mathcal{G}_{\pi_j} \cap G|^{\pi_j}$.*

Proof. Given the template matching criteria $|\mathbf{x}_1^g \wedge \mathbf{w}_1^g| / |\mathbf{x}_1^g| \geq \rho_1^g$ for $\mathcal{G}_{\pi_j}$,it is straightforward that the norm $|\mathbf{x}_1^g \wedge \mathbf{w}_{1j}^g|$ to make π_j to be selected is proportional to ρ_1^g, and $|\mathbf{x}_1^g \wedge \mathbf{w}_{1j}^g| \propto |\mathcal{G}_{\pi_j} \cap G|$. □

By reducing the goal vigilance ρ_1^g, an applicable plan π_j can still be selected although only a lesser number of propositions can satisfy the goal. The postcondition of π_j becomes the subgoal to be appended to the hypothetical sequence. The process continues iteratively and recursively to test each step of hypothetical action until the goal or the subgoal is achieved and new actions can be learnt.

Algorithm 1 illustrates the overall execution cycles of iFALCON. The basic goal monitoring, plan selection, plan execution, subgoaling, and backtracking can be depicted in the algorithm in line 3, lines 14–16, line 31–36, lines 32–34, and lines 7–10 respectively. Failure detection, hypothesis generation, and plan capture processes can be depicted in line 17, lines 18–29, and lines 4–6 respectively. The algorithm shows that the execution of the plan, the search for a novel solution, and the learning to capture the new plan can be interleaved under the main execution loop of the system execution. The critic attribute ($\mathbf{w}_j^c$) is modified based on the critic (matching) value (line 6 and line 18). A high critic value ($m_g > 0$) indicates that the plan is just created for learning or failed to achieve the goal.

Algorithm 1. iFALCON execution cycles.

1 **WHILE** ***True***
2 *Perceive the environment and update* F_1^b
3 IF $m_g = \dfrac{|\mathbf{x}_1^g \wedge \mathbf{x}_1^b|}{|\mathbf{x}_1^g|} \geq \rho^d$ /* the goal matches with the beliefs */
4 IF $\mathbf{max}(\mathbf{y}_3^s) \geq 1$ /* a new sequence has just been formed */
5 $\mathbf{w}_j^{t(\text{new})} = (1-\beta_3)\mathbf{w}_j^{t(\text{old})} + \beta_3(\mathbf{y}_3^s)$ /* learn the sequence */
6 $\mathbf{w}_j^{c(\text{new})} = (1-\beta_1^c)\mathbf{w}_j^{c(\text{old})} + \beta_1^c(\mathbf{x}_1^c)$ /* learn the critic */
7 IF $\mathbf{max}(\mathbf{y}_4^i) > 0$ /* some plans are pending achievements */
8 *readout* F_2 *and* F_3 *from node i* (**max**) *in* F_4; *reset node i*
10 *readout* F_1^g *from node j* (**max**) *in* F_2
11 ELSE ***Finish***
12 ELSE
13 IF $\mathbf{max}(\mathbf{y}_2^p) \leq 0$ /* no plan is activated or selected */
14 *select* F_2 *node j by resonance search*
15 $y_{2.j}^p \leftarrow 1; y_{2.m}^p \leftarrow 0,\ m \neq j$
16 *readout* F_1^g *and* F_3 *from node j in* F_2
17 IF $\mathbf{max}(\mathbf{y}_3^s) \leq 0$ /* no existing plan can be found */
18 $\mathbf{w}_j^{c(\text{new})} = (1-\beta_1^c)\mathbf{w}_j^{c(\text{old})} + \beta_1^c(\mathbf{x}_1^c)$ /* learn the critic */
19 *select* F_4 *node i by resonance search* (*recency gradient*)
20 REPEAT /* the start of the forward chaining hypothesis */
21 $\rho_1^{g(new)} = \rho_1^{g(old)} - \delta$ /* gradually reducing ρ_1^g */
22 *select* F_2 *node j by resonance search*
23 UNTIL *existing j is found*
24 *readout* F_1^g *and* F_3 *from* F_2 *node j* (**max**)

set action pattern in F_1^a to subgoal (achieve) action
*readout F_2 and F_3 from node i (**max**) in F_4*
resonance search F_3 with F_1^g and F_1^a(primacy gradient)
$\mathbf{w}_i^{s(new)} = (1-\beta_i^s)\mathbf{w}_i^{s(old)} + \beta_i^s(\mathbf{y}_3^s); \mathbf{y}_2^p = \mathbf{0}$
/* store the hypothesis; reset plan field */
set ρ_1^g back to normal (default)
ELSE /* there is an action in F_3 pending execution */
*readout F_1^a from F_3 nodet (**max**)*
IF *the selected action in F_1^a is a subgoal action (achieve)*
select F_4 node i by resonance search (recency gradient)
readout F_1^g from node t in F_3; reset F_2
reset node t in F_3
Execute the action based on F_1^a

To illustrate the process, Fig. 9 shows the trace of execution with the features of hypothesis generation and plan capture. The task use is the same as the one shown in Fig. 8. However, the plan shown in Fig. 7 is not available. Instead, some primitives are provided as follows:

```
{goal: [B_On_Table],          {goal: [A_On_Table],
pre: [-B_On_Table,            pre: [-A_On_Table,
Clear_B],                     Clear_B],
body: [Putdown_B]}            body: [Putdown_A]}

{goal: [A_On_B],              {goal: [B_On_A],
pre: [A_On_Table,             pre: [B_On_Table, Clear A],
Clear_B],
body: [Stack_A_On_B]}         body: [Stack_B_On_A]}
```

Following Algorithm 1, six execution cycles are required to solve the blocks world to reach the goal state from the initial state s_1 as shown in Fig. 5. The trace includes the hypothesis generation by reducing the vigilance parameter of ρ_1^g to find any intermediate step. The example also shows how the planning episode can be captured as a new plan. In the example, the process involving the critic is omitted for simplicity. Hypothesis generation starts whenever no plan can found to achieve the given goal (steps 3–8 in Fig. 9(i) and steps 2–8 in Fig. 9(iv)). The new plan can be learnt if the goal achievement still leaves the complete sequence activation in the sequencer (step 5 in Fig. 9(vi)).

4. Case study: blocks world

To test the model, iFALCON has been applied to the *blocks world* domain extended from the one used as an example in the previous section. Here, three blocks are used instead of two. To evaluate the model, two types of plan are applied as initial plans: primitives and a control plan. A primitive is a plan consists of only a single step of action as a basic rule that model the environment. A control plan is a more complex strategy with a sequence of several actions to achieve the goal. There are twelve different primitives and one control plan used in the experiment. The propositions used in the domain are also simplified. Propositions in the form of *x*_On_*y* are omitted. Instead, they are substituted with the equivalent more basic propositions like `Clear_`*x* or `On_Table_`*y*. For example, with three blocks A, B, and C, the proposition `A_On_B` can be substituted with a conjunction of several basic propositions such that $\texttt{AOnB} \Leftrightarrow (\texttt{ClearA} \wedge \neg\texttt{AOnTable} \wedge \neg\texttt{ClearB} \wedge \texttt{COnTable})$. This simplification can reduce the number of input nodes and neural connections to make it scalable without changing the meaning of the expressions.

4.1. Testing configuration

To evaluate the model, two cases are formulated as follows: (1) a blocks world planner with complete pre-given plans (both primitive and control plans) and (2) a planner with learnt incomplete plans (primitives only). Each configuration is tested

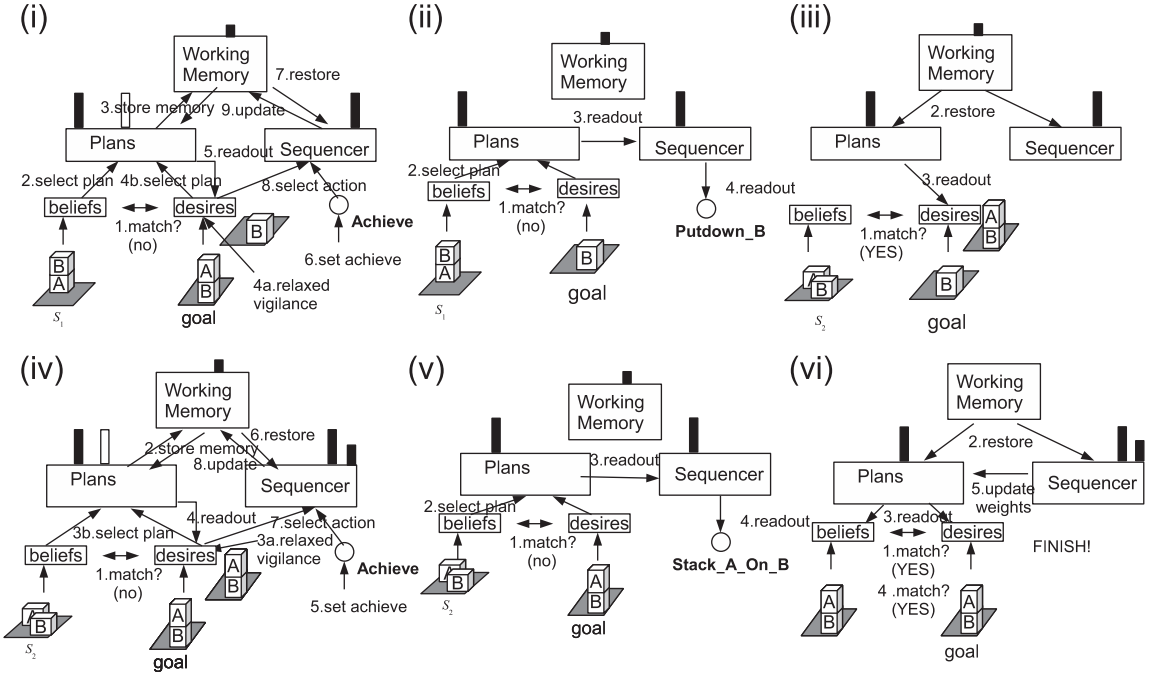

Fig. 9. The execution trace of solving a simple blocks world problem based on the hierarchical planning with search and learning algorithm of iFALCON. Six cycles of execution are required to achieve the goal from the initial state given only primitive actions. At the end, a new plan is learnt with two consecutive subgoals.

to achieve the goal from twelve different initial blocks configurations (Fig. 10). The control plan used in the experiment can be described symbolically as follows:

```
{goal: [Clear_A, -A_On_Table, -Clear_B,
-B_On_Table, -Clear_C, C_On_Table],
pre: [],
body: [{achieve: [Clear_C, C_On_Table]}, {achieve:
[-Clear_C, -B_On_Table]},
  {achieve: [-Clear_B, -A_On_Table]}]}
```

Each iFALCON plans configuration is applied to reach the goal from every block configuration. Different configurations may produce different numbers of steps. A random choice mechanism is applied to the internal mechanism of decision so that when more than one node have the same maximum values, a plan node is selected at random. Consequently, the choices may be different even for the same configuration.

The network parameters α, γ, β, and ρ for all fields are set to 1 except for $\rho_1^c = 0.8$ and $\beta_1^b = 0$ allowing the plan selection to select previously failed plan but still close to the main goal and keeping the existing precondition to the same values for the entire process. The increment (decrement) factor ϵ for both τ of F_3 and F_4 are 0.1. $\rho^e = 0.85$ for goal monitoring.

The test looks at the use of plan execution with the state-space search planning but without the online plan capture capability. For each block configuration and two different types of initial plans, the architecture is tested to achieve the goal configuration. Fig. 11 shows the average number of steps for each initial blocks configuration from 100 independent trials with no plan capture capability. The figure also shows the number of failure from 100 trials in the planning configuration in which only primitives are pre-given. The graph also shows the maximum and minimum steps taken for each configuration. The result shows that when control plans are available all blocks configuration can be solved for every trial. Few configurations require relatively many steps (configurations 7 and 4) with some degree of variability. This indicates that in certain cases, the control plan is not sufficient and the planner must search for alternative solutions. Using the complete set of plans, the goal can be achieved in three steps at minimum.

On the other hand, when only primitives are available, some configurations may fail (configurations 3, 5, 6, and 7). However, in most of the trials they eventually can reach the solution even though so many steps are taken. In 100 trials, maximally five are failed (in blocks configuration 6). In a case where the solution is straightforward (blocks configuration 8), it takes only a single step of action in primitives only planner instead of three like in the complete plans configuration which indicate that the control plan used is not optimal for all initial configurations. The result shows that iFALCON as a neural network architecture can be used as a hierarchical planning system that finds solutions even though the knowledge is incomplete.

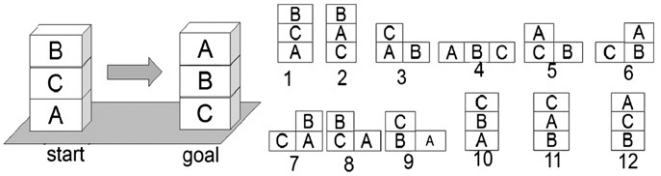

Fig. 10. The goal and different initial blocks configurations.

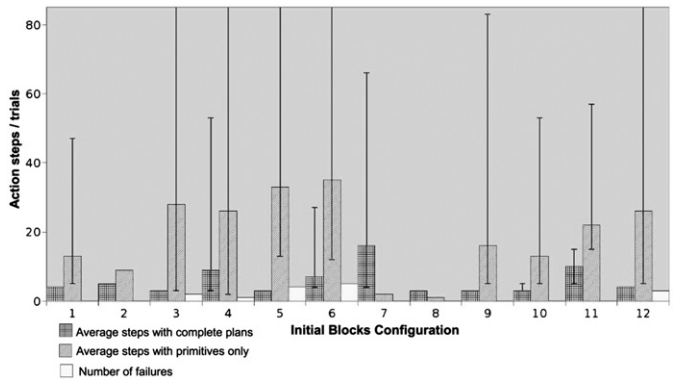

Fig. 11. Average number of steps and failures from 100 independent planning trails with complete plans and primitives only without plan capture capability.

4.2. Results and further tests

To evaluate the learning capability, another test is conducted using consecutive trials (plans learnt from previous trials are retained) to see whether it learns plans that can be useful to subsequent runs within the same configuration. Table 1 presents the results from 30 runs of 30 consecutive trials for each block configuration and each type of initial plans. The table shows the maximum and minimum number of steps from 30 consecutive trials that successfully achieve the goal. It also shows whether a performance improvement is indicated during the consecutive trials. A configuration is marked as 'indicated' if there is an increase or convergence of performance (reduction of the number of steps to achieve the goal) in consecutive trials. If there is an indication of improvement, the table also shows the maximum reduction or difference in the improvement. The number of achievement failures occur in 30 runs is also presented for each configuration.

The experiment reveals that when the control plan is included, most block configurations can be solved. There are cases in two configurations that the agent still fails to achieve (12 cases in configuration 6 and 8 cases in configuration 11). The failures can still happen due to the initial plans that are actually unsuitable when applied to those particular configurations.

However, the two configurations reveal that performance improvements can be made by learning new plans. In configuration 6, there is a case in which the number of steps is reduced by five after learning is conducted in a previous trial. In configuration 11 the number of steps is reduced by seven. The improvement indication, however, does not imply that the learning does not take place in other configurations. In fact, most configurations that have the same number of steps to achieve the goal also have cases involving plan capture at the first trial. The learnt plans makes the agent follows the same plans across different consecutive trials.

In one case, when the complete plans are applied as the initial configuration, the first trial can capture a new plan which can be mapped into its symbolic description as follows:

```
{goal: [C_On_Table, Clear_C],
pre: [-B_On_Table, -C_On_Table, A_On_Table, -
Clear_C, Clear-B, -Clear-A],
body: [{achieve: [B_On_Table, Clear_C]}, {achieve:
[C_On_Table, Clear-A]}],
critic: [1]}
```

The sample learnt plan above indicates that the plan is created to achieve a subgoal in the control plan. It implies that the

Table 1
Results of blocks world with learning.

Initial block conf.	With prior primitives + control plans					With prior primitives plans only				
	Number of steps		Improvement indicated		Failures	Number of steps		Improvement indicated		Failures
	Min	Max	Improved	Max diff.		Min	Max	Improved	Max diff.	
1	4	4				5	12			
2	5	5				9	9			
3	3	3								30
4	2	2								30
5	3	3								30
6	3	8	Indicated	5	12					30
7	2	2								30
8	3	2				1	1			
9	3	3				3	11	Indicated	4	28
10	3	3								30
11	4	11	Indicated	7	8					30
12	4	4				5	31			10

learning takes place only when necessary to plan or solve certain tasks. The learnt plan is automatically assigned with a critic attribute indicating that it can successfully achieve the goal.

On the other hand, when only primitives are provided initially, the result is not as good as when the complete plans are provided. Table 1 reveals that only 3 (configurations 1, 2, and 8) out of 12 configurations successfully achieve the goal without failures. Moreover, there are seven configurations that totally cannot be solved. Only few cases in one configuration (configuration 9) indicate performance improvement. Interestingly, there is a configuration in which the application of merely primitive plans becomes superior such as configuration 8. It takes only one step to solve the problem compared with the minimum three steps by its complete plans counterpart for the same configuration.

When only primitives are provided, one trial captures a new plan which can be shown in a symbolic description as follows:

```
{goal: [-B_On_Table, C_On_Table, -A_On_Table,
-Clear_C, -Clear_B, Clear-A],
pre: [-B_On_Table, -C_On_Table, A_On_Table,
-Clear_C, Clear_B, -Clear_A],
body: [{achieve: [Clear_A, B_On_Table]}, {achieve:
[-B_On_Table, -Clear_C]},
  {achieve: [-A_On_Table, -Clear_B]}],
critic: [1]}
```

The learnt plan above indicates that a reasonable plan can be learnt by chance from a set of primitives. The resulting plan above is also comparable to the control plan used in this experiment.

To confirm that the agent actually learns, some improvement must be indicated when the system runs to do the tasks over time. In that case, another experiment is conducted to reveal the characteristics of learning in iFALCON. The experiment looks at how well it achieves the goal continuously from a series of different varying initial block configurations (12 possible configurations) using the complete set of plans configuration.

In a single problem solving episode, a block configuration is selected at random. The experiment runs 20 independent trials over 500 consecutive series of problem solving episodes. For each trial, four values are measured:

- Number of execution cycles required to achieve the goal in a single episode. This includes both the number of action steps and the number of processing cycles to do with deliberation, subgoaling, and backtracking.
- Number of action steps taken to achieve the goal which includes only the action that changes the state of the environment (the state of the blocks) in a single episode.
- The difference between the number of action steps taken with the optimal number of steps that can be applied for that particular block configuration in a single episode.
- The number of plans learnt in a single episode.

Fig. 12(i) shows the trends of the average number of execution cycles and action steps taken in the overall series of episodes. Fig. 12(ii) shows the trends of the average difference between the number of action step and the optimal action possible. It also shows the average number of plans learnt in the overall series of episodes.

Fig. 12(i) and (ii) indicates that in continuous series of episodes of 12 randomly selected different configurations, the performance of the network does not converge. The trace of execution reveals that the source of the problem is in the failures of solving block configuration 11 which also indicated in the previous results. When the system fails to achieve the goal in one episode starting with block configuration 11, some plans learnt may deteriorate the performance of achieving the goal in other episodes. For example a plan $\pi_{j'}$, created in an episode initiated by configuration 11 can be as follows:

```
{pre: [-A_On_Table, -Clear_A, -B_On_Table, Clear_B,
C_On_Table, -Clear_C],
body: [{achieve: [Clear_A, B_On_Table]}, {achieve:
[-A_On_Table, -Clear_B]}],
goal: [-A_On_Table, -Clear_B],
critic: [1]}
```

The learnt plan above can lead the agent to take wrong steps. In block configuration 1, (initially `(-A_On_Table, -Clear_A, -B_On_Table, Clear_B, 'C_On_Table', '-Clear_C')`), the learnt plan above will be selected at the beginning instead of the control plan. This can happen as the main goal G still match with the plan precondition $\mathcal{G}_{\pi_{j'}}$ such that $G \supseteq \mathcal{G}_{\pi_{j'}}$. Consequently, the main goal will never be achieved even though $\mathcal{G}_{\pi_{j'}}$ is satisfied. This faulty plan influences all episodes that begin with some configuration that match with the precondition and the goal of the plan. This condition also reveals one of the weaknesses of the current hypothesis generation step in iFALCON in which the subgoal generated are still bounded to the postcondition of an

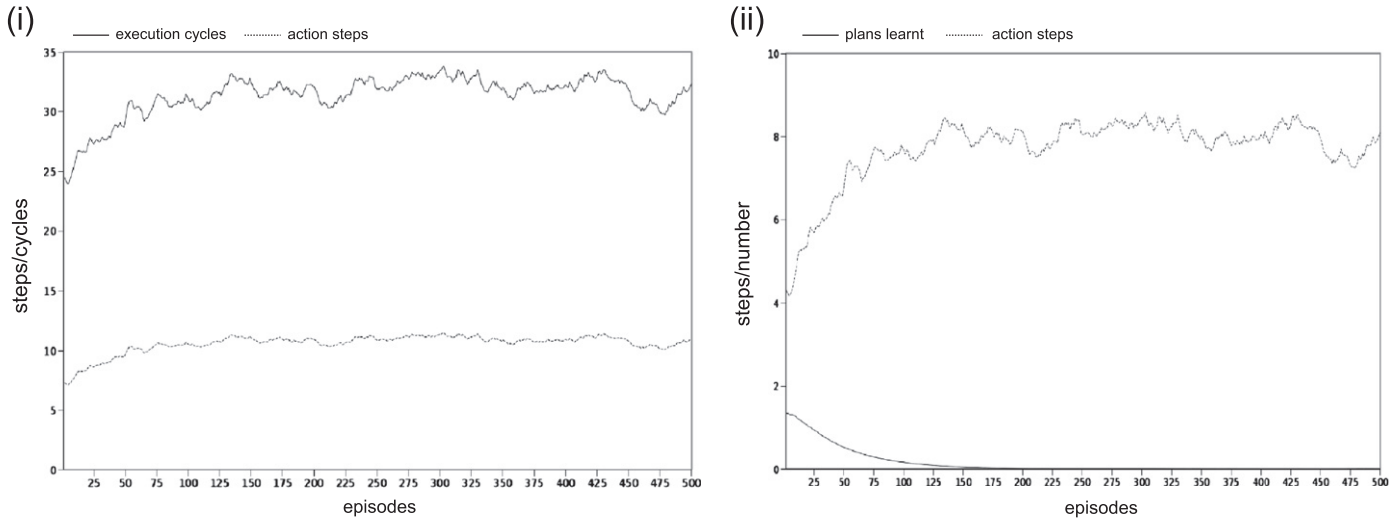

Fig. 12. (i) The trend of average over 20 trials of the number of execution cycles and action steps needed to solve a problem for 500 episodes and (ii) the trend of average over 20 trials of the average difference between the number of action step and the number of optimal actions possible, and the average number of plans learnt in the overall series of episodes for 500 episodes.

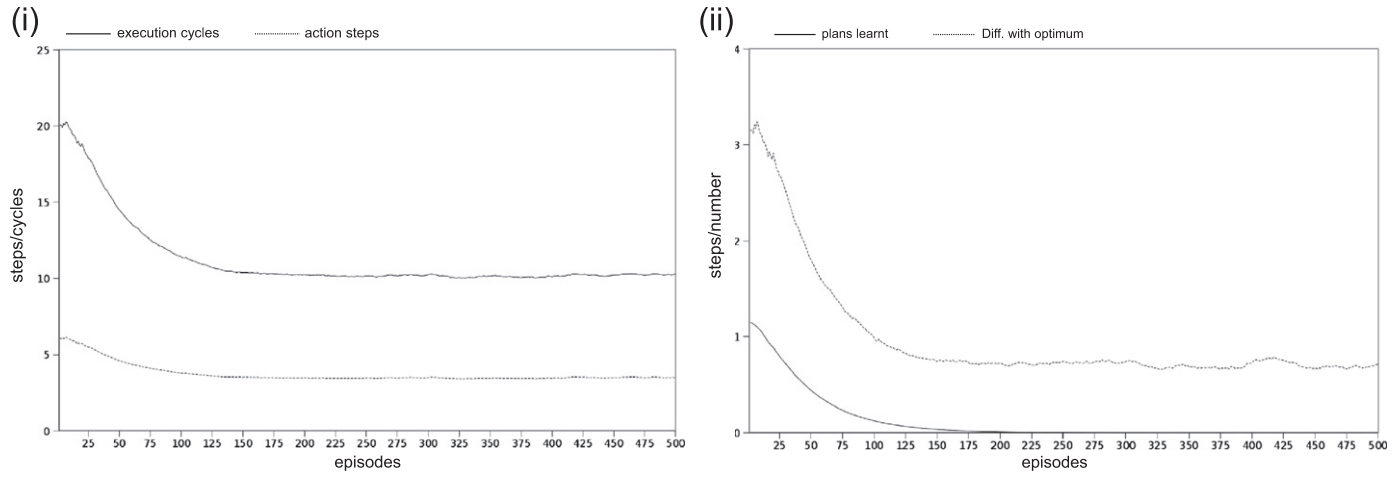

Fig. 13. (i) The trend of average over 20 trials of the number of execution cycles and action steps needed to solve a problem for 500 episodes (excluding block configuration 11) and (ii) the trend of average over 20 trials of the average difference between the number of action step and the optimal action possible, and the average number of plans learnt in the overall series of episodes for 500 episodes (excluding block configuration 11).

pre-existing plan. A faulty initial plan description may impair the goal monitoring process.

When the configuration 11 is taken out from the list of initial block configuration, the performance improves and converges. Fig. 13(i) shows the trends of the average number of execution cycles and action steps without block configuration 11. Fig. 13(ii) shows the trends of the average difference between the number of action step and the optimal action possible, and the average number of plans learnt without block configuration 11. Fig. 13(i) and (ii) shows that all tested aspects improve over 500 episodes starting from 11 initial block configurations. Both number of action steps taken and the execution cycles (including steps taken for reasoning) are decreased and converged after about 125 consecutive episodes. However, the solutions in average can still not reach the optimal (in average, it eventually stays at one step difference with the optimal solution). These less than optimal solutions are found because the control plan used as the main guide for execution and learning is actually not optimal for all blocks configuration. This is indicated in the result that after about 200 consecutive episodes, no more learning is conducted although the average number of steps is still greater than the optimal.

The results indicate that the quality of planning and learning is sensitive to the pre-existing knowledge and the initial task configuration. A wrong choice to start with may also lead the agent to further learn the inappropriate knowledge to solve the task. However, the experiment also shows that iFALCON can learn plans as explicit descriptions and execute them accordingly. It also clearly indicates that it plans and learns at the same time when the knowledge to solve a particular problem is not available.

5. Case study: Unreal Tournament NPC

We have also applied iFALCON in a more realistic real time situation in which the planning architecture is used as a part of a non-player character (NPC) agent inside the Unreal Tournament (UT) realtime first-person shooter video game [18]. The main objective of implementing this second domain is to simply

demonstrate that iFALCON can execute and capture plans on the run when the environment is continuous and dynamic. Given some initial plans which may be incomplete, we investigate whether an iFALCON agent can capture more plans from experiences which may be useful for the agent to accomplish its mission. Compared with the previous domain of blocks world, this domain requires a planning agent to run continuously to accomplish its tasks while learning new plans. On the other hand, blocks world employs discrete tasks or episodes of problem solving.

In UT, the NPC agent is resided in a 3D virtual environment where there are some objects to collect that may be useful for its task. There can be weapons, ammunitions, armours, health boosters and other objects that the agent can discover and collect while being engaged in a battle situation with other NPCs or players using weapons to attack or defend.

5.1. Domain modeling

In this UT domain, the NPC agent is made to accomplish a mission which may involve following a sequence of objectives. The mission applied is to continuously collecting certain objects resided in the environment which must be done in a certain order. The mission is given explicitly as goals and plans inserted as neural activations and connections. This simple task is set simply to test that the architecture can work as both both plan executor and plan generator at the same time.

The particular mission employed as a test case is to let the NPC agent to collect a flak cannon (weapon) first followed by collecting a rocket launcher while it may be engaged in a battle situation with other NPCs. The cycle of weapons collection continues after the second target (collect the rocket launcher) is achieved. The mission iteration can be made possible because an item collected inside the UT environment will be re-spawned in the same location for several seconds after being collected. Based on the same goals and the context, the mission will eventually be re-enacted. To navigate around the environment, the NPC can travel from one navigation point to another. A navigation point is an pre-existing landmark in the environment that the NPC can sense and locate.

Based on the information provided by UT and the particular mission setup, there are 17 propositions that can be used to characterize the beliefs or perception of the agent about its surroundings and itself. Each proposition can have either a binary value or a real value between 0 and 1. Table 2 shows the list of propositions and their descriptions. All propositions in the table are binary valued except `healthLevel` and those in the last row. `sStraight`, `sUp`, `sDown`, `sLeft`, and `sRight` are real valued between zero and one.

The values of the propositions can be mapped into a vector and becomes the input pattern for the F_1^b field of iFALCON. The input field employs *complement coding* so that each proposition is represented as a pair of values allowing *do not-care* condition. Some values of propositions can be updated accordingly whenever there is a change in the environment. However, some propositions do not have direct correspondences to properties of the environment. `missionAcc`, `gotLauncher`, `gotFlak`, and `toMove` indicate some instant values that hold only for a short time. For example, the condition `gotLauncher` can only be true when the agent just picks up the rocket launcher but the value only lasts momentarily. To deal with it, those propositions or values are held a bit longer in a memory buffer when the corresponding messages received so that the iFALCON cycle can still have time to process the values.

Besides perception, the vector of the action field is also mapped into a command that can be sent out for execution. The action field is competitive so that only a single node is active at one time. Table 3 shows the corresponding propositions of the action field and their descriptions (`achieve` action for subgoaling is omitted from the table for simplification).

Based on the structure of input/output fields of iFALCON for UT NPC domain, it is indicated that initially the agent does not have any knowledge nor representation about locations or positions of objects and itself in the environment. The experiment looks at whether iFALCON can generate and capture plans reflecting positions of objects relative to the agent and the task at hand.

5.2. Testing configuration

Initially, some predefined plans are inserted into the network. The prior plans in symbolic forms can be described in Table 4. The initial plans show that the agent will try to get a flak cannon followed by a rocket launcher in order to accomplish the mission (`missionAcc`). But it is also shown that the plans are still incomplete. There is no predefined plan for achieving the condition that a rocket launcher nor a flak cannon is reachable. Thus, although the agent knows how to move to the launcher or the cannon, it still must figure out how to make them reachable while it tries to accomplish the mission. The agent may be building up a certain form of map consisting of plans.

Similar to the blocks world domain, the network parameters α, γ, β, and ρ in this domains are set to 1 except $\rho_1^c = 0.8$, $\beta_1^b = 0$, and $\epsilon = 0.1$. However, $\rho^e = 0.95$ is used to make the goal monitoring less tolerating as each goal proposition is characterized by a specific

Table 2
Perception proposition symbols.

Proposition	Description
`missionAcc`	Signals the completion of the NPC mission
`gotLauncher`	Signals the NPC that a rocket launcher is just obtained
`gotFlak`	Signals the NPC that a flak cannon is just obtained
`farWeapon`	Indicates whether a weapon is reachable or not
`farHealth`	Indicates whether a healthkit is reachable or not
`farCollect`	Indicates whether another type of collectible item is reachable or not
`farEnemy`	Indicates whether another player is seen or not
`farNavpoint`	Indicates whether there is a reachable navigation point
`farLauncher`	Indicates whether a rocket launcher is reachable or not
`farFlak`	Indicates whether a flak cannon is reachable or not
`healthLevel`	Indicates the bot's health level
`toMove`	Signals the bot when the bot has just arrived at a navigation point
`sStraight, sUp, sDown, sLeft, sRight`	Distance sensor values (between 0 and 1) for five corresponding directions

Table 3
Action proposition symbols.

Proposition	Description
mov2Navpoint	Triggers the bot to move to another reachable predefined navigation point
lookforNavLeft	Starts the bot to rotate to the left until it finds any reachable navigation point
lookforNavRight	Is the same as lookforNavLeft but to the right direction
move2Weapon	Triggers the bot to move to and collect a reachable weapon
move2Health	Triggers the bot to move to and collect a reachable healthkit
move2Collect	Triggers the bot to move to and collect any other reachable collectible item
move2Launcher	Triggers the bot to move to and collect a reachable rocket launcher
move2Flak	Triggers the bot to move to and collect a reachable flak cannon
shootEnemy	Triggers the bot to shoot any player seen by the bot

Table 4
Prior plans in the UT NPC.

```
{goal: [gotFlak],
pre: [-gotFlak],
body: [{achieve:[-farFlak]},
  {do:[move2Flak]}],
critic: [1.0]},

{goal:[gotLauncher],
pre:[gotFlak],
body:[{achieve: [-farLauncher]},
  {'do':['move2Launcher']}],
critic:[1.0]},

{goal: [toMove],
pre:[-farNavpoint],
body: [{do:[move2Navpoint]}],
critic: [1.0]},

{goal: [-farNavpoint],
pre: [farNavpoint],
body: [{do: [lookforNavRight]}],
  critic: [1.0]}

{goal:[-farNavpoint],
pre: [farNavpoint],
body: [{do:[lookforNavLeft]}],
critic: [1.0]},

{goal: [farEnemy],
pre:[-farEnemy,healthLevel=1.0],
body:[{do:[shootEnemy]}],
util:[1.0]},

{goal:[farEnemy],
pre:[-farHealth],
body: [{do: [move2Health]}],
util: [1.0]},

{goal:[missionAcc],
pre: [],
body: [{achieve: [gotFlak]},
  {achieve: [gotLauncher]}],
    util: [1.0]}
```

event or signal. iFALCON is evaluated under continuous realtime situation. However, unlike the previous domain, this UT Gamebot agent is only tested for its ability to execute plans and the possibility that any plan can be captured as a result of the planning and learning capability driven by the algorithm. The overall performance or improvement as produced by learning is not evaluated.

In that case, the UT NPC agent is tested in a single instance of the game only. The bot runs to accomplish its task in a UT game instance for about 15 min while the number of items collected and the plans captured are recorded.

5.3. Results

After 824 ticks (1 tick is 100 interaction cycles or about 1 s) the agent can accomplish the mission 6 times, and get 13 items in the right order (get the flak cannon first followed by getting the rocket launcher). It is indicated that the agent has followed the initial plans as no false attempt is conducted (e.g. the agent tries to pick up another flak cannon though it is just taken previously).

The NPC has also captured many new plans although most of them consist of only a single step of action. Some plans produced are also redundant (a plan captured is very similar to another one already learnt previously) as the basic Algorithm 1 still does not have a mechanism to avoid redundancy. The high level vigilance ($\rho = 1$) for most input fields also contribute to the large number of plans created as some input attributes comprise real values. In this section we only show a few number of plans captured after cycles of runs.

Some plans captured that bridge the gap of the missing plans are as follows:

```
LEARNT PLAN 1:
{pre: [-missionAcc, -gotLauncher, -gotFlak,
farWeapon, farHealth,
  farCollect, farEnemy, -farNavpoint,
farLauncher, farFlak,
  healthLevel=1.0, -toMove, sStraight=1.0,
sUp=0.85,
  sDown=0.12, sLeft=1.0, sRight=0.98],
critic: [1.0],
body: [{achieve: [toMove]}, {achieve: [toMove]}],
goal: [-farFlak]
}

LEARNT PLAN 2:
{pre: [-missionAcc, -gotLauncher, gotFlak, -
farWeapon, farHealth,
  farCollect, farEnemy, -farNavpoint,
farLauncher, farFlak,
  healthLevel=1.0, -toMove, sStraight=0.78,
sUp=1.0,
  sDown=0.12, sLeft=0.89, sRight=1.0],
critic: [1.0],
body: [{achieve: [toMove]}, {achieve:
[-farNavpoint]},
  {achieve: [toMove]}, {achieve: [toMove]}],
```

```
goal: [-farLauncher]
}

LEARNT PLAN 3:
{pre: [-missionAcc, -gotLauncher, -gotFlak,
farWeapon, farHealth,
  farCollect, farEnemy, farNavpoint, farLauncher,
farFlak,
  healthLevel=1.0, -toMove, sStraight=1.0,
sUp=1.0,
  sDown=0.12, sLeft=1.0, sRight=1.0],
util: [1.0],
body: [{achieve: [-farNavpoint]}, {achieve:
[toMove]},
  {achieve: [toMove]}],
goal: [-farFlak]
}
```

From the tested game instance, the agent can learn sequences of subgoals ranging from one to four steps. The precondition of a plan characterizes the situation that the particular plan is applicable. The learnt symbolic plans above include attributes or propositions involving real values to represent sensor reading and health level in the plan preconditions (`sStraight, sUp, sDown, sLeft, sRight, healthLevel`). These real values are the features of the architecture that may not be available in its resembling symbolic models.

To some extent, the plans captured may be reasonable and consistent with the task domain. For example in the learnt plan 3 above, the sequence is started with a subgoal of finding a navigation point before performing a movement action (`toMove`) which is still make sense in that case.

Overall, the UT NPC case study has indicated that iFALCON can function as a plan executor and plan generator simultaneously in a continuous realtime domain. Despite the overall performance, the efficiency, and the consistency of the learnt plans, the architecture can capture plans that can be useful and meaningful if the domain model and the initial knowledge are also designed and set up appropriately.

6. Discussion

Some interesting aspects of the neural network model presented in this paper are the approach of realizing the state-space search mechanism and the representation of sequential or hierarchical relationships as a transient structure in the neural network. These features of planning are rarely addressed by neural networks. This paper has demonstrated that the approach can successfully realize a fully functional hierarchical planning that can learn by capturing planning episodes on the fly. However, as a planner, the model suggested in this paper is still limited in some ways. At this stage, the correctness and completeness criteria of the plan discovered are still omitted. As mentioned in the previous section, there are some conditions in the case study that can lead the planning to dead-ends, especially when the hypothesis generation and the plan capture are involved. There are several possible solutions to this condition which will be discussed below.

An important capability for a planning system is to detect a state that leads to a dead end. A common approach in planning is to identify a circular situation in which one action may lead to a state that has been visited previously during the planning. In the current model of iFALCON, this capability has been addressed partially in which plans that have been selected previously will simply not be selected in the next round of the resonance search. However, the effectiveness of this approach depends on the complexity of the problem. As all applicable plans have been selected, the filter is simply refreshed and all plans become unrestricted for selection so that the problem remains. An alternative solution is to check whether the same subgoal action has been selected in the sequencer field F_3 during the subgoal insertion in the plan capture cycle. This can be realized as a part of the resonance search to select and activate a subgoal action to capture the subplan. If the same subgoal has been selected before, the search can resume to find another plan.

The approach of hindering the circular situations may not be the most effective method to deal with the complexity of the problem. The current adopted naive forward chaining method in the plan search may not be suitable to handle large branching options towards the solution. Based on the current model and the algorithm of iFALCON, it is also possible to apply a backward chaining search method besides the basic forward one. As opposed to the forward chaining search, the beliefs field F_1^b can be gradually decreased to find an alternative plan so that the search moves backward from the goal to the possible plan that can make the goal achievable. Instead of directly replay the selected alternative plan like in the forward method, the backward search just stores the selected alternative plan if any to the working memory and replaces the beliefs field F_1^b with the content of desires (F_1^g). This approach requires the perception input to be blocked temporarily for the next round of the execution cycle. In this way, it is not only just backward-chaining can be realized in iFALCON, but it can also be combined with forward-chaining method so that if no plan that can directly achieve the goal is available, the forward-chaining search takes over to just select any plan that is applicable.

Another limitation in the current iFALCON model of learning is that generalized postconditions in pre-existing plans constrain postconditions of the plans learnt or captured. This condition may impair further learning processes in a longer term as indicated in the results of the blocks world experiment. One way to solve this problem is to let the hypothetical subgoal and the postcondition of the learnt plan to be made more specific by merging the values of the readout postcondition and the current perceived situation. Further learning process can also be made to generalize the postcondition through the template learning process.

Another interesting aspect of iFALCON is the use of critic field. Beyond the similarity measure between the goal and perception, an interesting improvement is to extend the critic field to include different values like cost, length, risk, reward and even emotional factor. This extension may integrate some cognitive aspects to the planning domain.

7. Conclusion

This paper has presented a model of hierarchical planning system realized as a self-organizing neural network architecture. The model explains how plans can be mapped into a composition of multi-channel adaptive resonance theory networks. The model uses a new kind of temporal activation encoding to represent sequences and to handle hierarchical processing structure. The encoding technique allows sequences to be grouped and processed resembling the structure and operations of plans. The model emulates the processes involved in an intentional agent architecture by seamlessly integrating deliberation and plan execution as a single unit of activation cycles. Furthermore, based on the principle of adaptive resonance theory, planning and learning can be integrated as parts of the activation cycles.

The neural planning model has been implemented and tested to solve the blocks world problem. Beyond a static plan-based executor, the experiment confirms the capability of planning and learning to explore and capture new solutions. However, the test also reveals that the quality of planning and learning is sensitive to the availability of the appropriate prior knowledge. More

in-depth studies are required to obtain the complete picture of the characteristics of the plan learning so that initial plans and network parameters can be setup more effectively.

Some aspects of the model are not yet fully exploited at the current stage. The application of the critic attribute of the plan and more general mechanisms to handle options and alternative plans deserve completion so that the model can be improved to deal with more different types of domain and environment. The experiments conducted with the current implementation are still limited to plans involving binary representation or just specific analog values. Moreover, the current problem domains are still not reflecting the generalization feature of the fusion ART process. The feature would enable iFALCON automatically identifies key features of the state and preconditions and simplifies the plan generated. It is also possible to learn a range of values besides a specific value using the fuzzy operations involved in the matching and learning process.

The experiment conducted has indicated the potential of integrating a myriad of reasoning and learning mechanisms for systems accomplishing certain tasks. The proposed neural model comprises bi-directional pathways of activations which make it possible to select a plan or activating a sequential pattern from different directions so that the plan can also be selected based on the presentation of action sequences rather than triggered by goals. Although the model and the experiment are still designed for a single agent domain, it is possible to extend the model to deal with more complex issues involving multiple agents like plan recognition or learning by imitation. In the future, it is also possible to build a society of neural network agents employing and constructing social norms.

In any case, the proposed model can bridge two different approaches of building planning agents. Top-down formal symbolic approaches can be integrated with bottom-up non-symbolic processes to accomplish a single task domain. Both directions can support and enrich each other to realize a system that ultimately deliberate, plan, and learn.

Acknowledgement

This research is supported by the Singapore National Research Foundation & Interactive Digital Media R&D Program Office, MDA, under research grant NRF2008IDM-IDM004-037.

References

[1] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufman, Amsterdam, 2004.
[2] E.D. Sacerdoti, The nonlinear nature of plans, in: Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75), 1975, pp. 206–218.
[3] L. Spalazzi, A survey on case-based planning, Artif. Intell. Rev. 16 (1) (2001) 3–36.
[4] T. Zimmerman, S. Kambhampati, Learning-assisted automated planning: looking back, taking stock, going forward, AI Mag. 24 (2) (2003) 73–96.
[5] O. Ilghami, D.S. Nau, H. Munoz-Avila, D.W. Aha, Learning preconditions for planning from plan traces and htn structure, Comput. Intell. 4 (2005) 388–413.
[6] C. Hogg, H. Munoz-Avila, U. Kuter, HTN-MAKER: learning HTNs with minimal additional knowledge engineering, in: Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence, 2008.
[7] C. Hogg, U. Kuter, H. Munoz-Avila, Learning hierarchical task networks for nondeterministic planning domains, in: IJCAI-09, 2009.
[8] X. Wang, Planning while learning operators, in: Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, AAAI Press, 1996, pp. 229–236.
[9] S.W. Bennet, G.F. Dejong, Real-world robotics: learning to plan for robust execution, Mach. Learn. 23 (2–3) (1996) 121–161.
[10] M. Beetz, Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures, Lecture Notes in Computer Science, vol. 1772, Springer, Berlin, 2000.
[11] R. Sun, Duality of the Mind: A Bottom-up Approach Toward Cognition, Lawrence Erlbaum, Mahwah, 2002.
[12] G. Baldassarre, A modular neural-network model of the basal ganglia's role in learning and selecting motor behaviours, J. Cogn. Syst. Res. 3 (2002) 5–13.
[13] L. Shastri, D.J. Grannes, S. Narayanan, J.A. Feldman, A connectionist encoding of parameterized schemas and reactive plans, in: G. Kraetzschmar, G. Palm (Eds.), Hybrid Information Processing in Adaptive Autonomous Vehicles, Springer Verlag, Berlin, 1997.
[14] M. Garagnani, L. Shastri, C. Wendelken, A connectionist model planning via back-chaining search, in: Proceedings of Cognitive Science 2002, Lawrence Erlbaum, Mahwah, 2002, pp. 345–350.
[15] A.-H. Tan, FALCON: a fusion architecture for learning, cognition, and learning, in: Proceedings, International Joint Conference on Neural Network (IJCNN'04), 2004, pp. 3297–3302.
[16] A.-H. Tan, G.A. Carpenter, S. Grossberg, Intelligence through interaction: towards a unified theory for learning, in: International Symposium on Neural Networks (ISNN) 2007, vol. 4491, Lecture Notes in Computer Sciences, Nanjing, China, 2007, pp. 1098–1107.
[17] A.-H. Tan, N. Lu, D. Xiao, Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback, in: IEEE Transactions on Neural Networks, vol. 9, 2008, pp. 230–244.
[18] D. Wang, B. Subagdja, A.-H. Tan, G.-W. Ng, Creating human-like autonomous players in real-time first person shooter computer games, in: Proceedings of the Twenty-first Annual Conference on Innovative Applications of Artificial Intelligence (IAAI'09), 2009, pp. 173–178.
[19] B. Bonet, H. Geffner, Planning as heuristic search, Artif. Intell. 129 (1–2) (2001) 5–33.
[20] J. Hoffman, B. Nebel, The FF planning system: fast plan generation through heuristic search, J. Artif. Intell. Res. 14 (2001) 253–302.
[21] M.E. Bratman, Intention, Plans and Practical Reason, Harvard University Press, Cambridge, 1987.
[22] M.E. Pollack, The uses of plans, Artif. Intell. 57 (1) (1992) 43–68.
[23] F. Ingrand, M. Georgeff, A. Rao, An architecture for real-time reasoning and system control, IEEE Expert 7 (6) (1992) 34–44.
[24] A.S. Rao, M.P. Georgeff, BDI agents: from theory to practice, in: Proceedings of the First International Conference on Multi-agent Systems (ICMAS-95), San Francisco, 1995.
[25] G.A. Carpenter, S. Grossberg, D.B. Rosen, Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system, Neural Networks 4 (1991) 759–771.
[26] G.A. Carpenter, S. Grossberg, Adaptive resonance theory, in: M. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA, 2003, pp. 87–90.
[27] S. Grossberg, L. Pearson, Laminar cortical dynamics of cognitive and motor working memory, sequence learning, and performance: toward a unified theory of how the cerebral cortex works, Psychol. Rev. 115 (2008) 677–732.
[28] S. Grossberg, Behavioral contrast in short-term memory: serial binary memory models or parallel continuous memory models? J. Math. Psychol. 3 (1978) 199–219.
[29] G. Bradski, G.A. Carpenter, S. Grossberg, STORE working memory networks for storage and recall of arbitrary temporal sequences, Biol. Cybern. 71 (1994) 469–480.

Budhitama Subagdja is a research fellow in School of Computer Engineering (SCE), Nanyang Technological University, Singapore. He received his PhD from the Department of Information Systems, the University of Melbourne, Australia. He finished and obtained his undergraduate and master degree in computer science from the Faculty of Computer Science, University of Indonesia. Before he joined NTU, he worked as a research assistant and a lecturer in the University of Indonesia. He was also a postdoctoral fellow at the University of Melbourne after finishing his PhD. His current research interests include planning, reasoning, and learning mechanisms in autonomous agents and biologically inspired cognitive architecture for intelligent agents.

Ah-Hwee Tan is an associate professor and the head of Division of Software and Information Systems at the School of Computer Engineering (SCE), Nanyang Technological University. He was the founding director of Emerging Research Laboratory, a research center for incubating new interdisciplinary research initiatives. Dr. Tan received a PhD in cognitive and neural systems from Boston University, a Bachelor of Science (First Class Honors) and a Master of Science in computer and information science from the National University of Singapore. Prior to joining NTU, he was a research manager at the A*STAR Institute for Infocomm Research (I^2R), spearheading the Text Mining and Intelligent Agents research programmes. His current research interests include brain-inspired intelligent agents, cognitive and neural systems, machine learning, knowledge discovery and text mining.