Advances in Cognitive Systems 2 (2012) 239-256 Submitted 9/2012; published 12/2012

Flow Maximization as a Guide to Optimizing Performance: A Computational Model

Vadim Bulitko BULITKO@UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, AB, T6G 2E8, Canada

Matthew Brown MBROWN2@UALBERTA.CA
Department of Psychiatry, University of Alberta, Edmonton, AB, T6G 2E8, Canada

Abstract

Flow is a psychological state linked to optimizing cognitive performance in humans. In this paper we propose a simple computational model of flow. We first define the degree of flow as the quality of the match between the agent's cognitive skills and the cognitive complexity of its task. In a hierarchy of increasingly more complex and rewarding tasks, taking on the task of a matching complexity allows the agent to maximize its performance. It also consequently maximizes the degree of flow the agent will experience. We take advantage of this connection and make our agents explicitly aware of the degree of flow they are experiencing. Maximizing the readings from such a "flow meter" improves the agent's ability to explore the environment and find problems of matching complexity. Thus, maximizing the degree of flow becomes a guide to maximizing the agent's performance in the environment. We implement these ideas by extending the standard value iteration learning method with planning and real-time operation and empirically demonstrate that flow-maximizing agents tend to collect more reward from the environment.

1. Introduction

The psychological condition of flow has been linked to optimizing cognitive performance in humans (Csikszentmihalyi, 1990). People experiencing the condition of flow appear to be fully engaged in their task, and their cognitive faculties and attention are fully focused on the task. In order for this to happen, there has to be a match between the task complexity and the individual's abilities. Tasks that are too simple or too complex for the individual cause other psychological states such as boredom or anxiety. Reaching the condition of flow has been linked to a number of benefits, including higher productivity and happiness. Hence, flow appears to optimize the application of cognitive abilities in humans. Thus, it is of interest to model flow mathematically as well as to consider whether giving artificial cognitive agents an ability to sense flow and the desire to maximize flow can optimize their performance too.

We consider environments comprised of tasks of different cognitive complexity. More complex tasks require a higher cognitive skill level to master but are also more rewarding. We then consider agents of different levels of cognitive skills. By matching the complexity of the task the agent takes on to its skills, the agent stands to improve its performance and collect the most reward out of the environment. We define the *degree of flow* an agent is experiencing in its current state as the quality of the match between the agent's skills and the complexity of the task the agent is attempting to solve. The better the match the higher the degree of flow. Thus, matching the agent's skill level and the task complexity not only improves the agent's performance in the environment but also increases the degree of flow the agent is experiencing.

We take advantage of this connection and postulate that some agents may be explicitly aware of the degree of flow they are experiencing and intentionally attempt to maximize it. They do so by taking on tasks whose cognitive complexity matches their cognitive skill level. Consequently, they improve not only their degree of flow but also their performance on the task. Thus, we can think of maximizing the degree flow as a *guide* to maximizing an agent's performance.

We implement these ideas within the reinforcement learning (RL) framework (Sutton & Barto, 1998). In the RL framework, an agent attempts to maximize the total reward it collects over its life time. Each reward is a scalar the agent receives from the environment immediately upon taking an action. We implement flow awareness by adding a second reward signal proportional to the degree of flow the agent is experiencing. Doing so allows RL agents to more quickly and robustly locate the task of a matching skill level and, as a result, collect more (non-flow) reward from the environment.

2. Related Work

Several computational models of flow have been proposed. For instance, the synchronization model of flow links the degree of flow with the amount of synchronization and connectivity between the attentional and reward networks in humans (Weber et al., 2009). Structural Equation Modeling was used to connect the degree of flow to other experiences of humans (e.g., time distortion and focus of attention) (Jin, 2012). These models appear to be specific to the brain faculties of humans and/or human-specific experiences. Thus, it is not immediately clear how to apply them to AI agents that may not share the same faculties or experiences.

In the field of AI, the research on intrinsically-motivated and curiosity-driven learning attempts to give agents a principled way to explore and learn the environment (Lim & Auer, 2012). Such research is relevant to flow insomuch as it provides a framework for driving the agent towards novel tasks. However, it does not explicitly define the notion of flow or seek to match the task complexity and the agent's cognitive abilities to increase the degree of flow experienced by the agent.

3. Our Model of Flow

We begin with the intuition for our model and then support it with formal details. For the sake of clarity and simplicity, we base the quality of the match between the cognitive complexity of a task and the agent's cognitive skill level on the *amount of planning time* the agent needs to expend per action to handle the task. Other kinds of cognitive resources (e.g., memory, the ability to abstract) are considered in the future work section.

3.1 Intuition

We consider an agent operating in an environment. The timeline is discrete, and on every time step the agent perceives the environment and forms its state. The agent decides on its action, which it then passes to the environment. The environment changes in response to the action, and a new time step begins. The process stops after an *a priori* fixed number of steps, at which point the agent dies. Our environment is real-time insomuch as the agent has a fixed time limit on its deliberation during each time step. If the agent exceeds the limit, it is punished with a negative reward. This represents the lost opportunity cost (e.g., not hitting a ball at the right time in tennis).

Return maximization. On every time step, the agent receives a scalar reward from the environment. The agent's objective is to optimize the cumulative reward collected over its life time. The agent estimates the potential for cumulative future reward (called "return") for different actions taken in different states. Then, given the current state, the agent takes the action that has the highest estimated return in that state. However, the agent's cognitive ability to represent expected returns of states is limited. This may be due to the use of function approximation methods which frequently over-generalize returns from one state onto another. Alternately, this may be due to state aliasing where the state features used to represent a state return do not distinguish one state from another.

Thus, depending on the complexity of the environment, the agent may not be able to represent the returns accurately. The agent uses lookahead (Korf, 1990) to compensate for the inaccuracy of the expected state returns. The agent *looks ahead* by imagining itself in future states and simulating the actions it can take in such states as well as the reward it would collect by taking the actions. This involves self-reflection as the agent needs to think about its own reactions (rewards) to hypothetical situations (imaginary states). The action with the highest expected value is then applied in the current state (Figure 1).

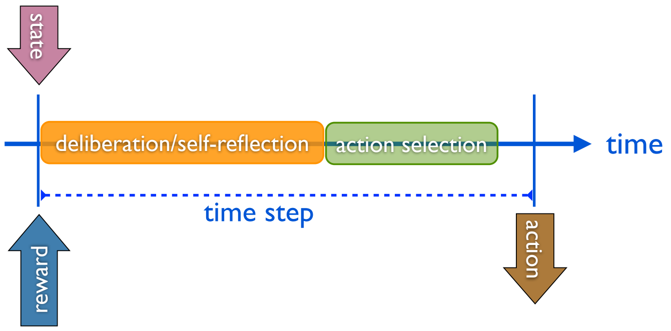


Figure 1. A single time step in the life of the agent. At the beginning of the time step, the agent collects reward from the previous time step and perceives its new state. It then conducts a lookahead, improves its return estimates and uses them to select the best action.

Lookahead. More time spent looking ahead tends to improve the accuracy of the return estimates and, consequently, the quality of the agent's actions. On the other hand, spending too much time looking ahead causes the agent to exceed the deliberation time limit, resulting in punishment with negative rewards. The amount of looking ahead is based on the agent's cognitive ability to learn and represent optimal returns for an area of the space. Specifically, if for certain states the agent is unable to learn and represent the returns accurately, then deeper lookahead is needed to compensate. Conversely, if it is possible to learn and represent the optimal returns more accurately, then the agent can rely on its learned representation of the returns and select the next action with little lookahead. In the limit, the agent will cease to look ahead and will choose its actions reflexively, based purely on its perfect estimates of the state-action returns. This means that the agent's cognitive ability to represent return estimates determines the maximum complexity of the task the agent can master. More complex environments require more lookahead than there is time available.

As proposed in the introduction, we define the degree of flow as the quality of the match between the agent's cognitive resources and the cognitive complexity of the task. Within the framework developed in this section such match quality is measured on the basis of per-action lookahead time as follows: the degree of flow the agent experiences in a time step is higher the more closely the deliberation time matches the duration of the time step without exceeding it.

Learning. In psychology, the match between task complexity and an agent's skill is a necessary condition for the state of flow. The quality of such a match is dynamic since the agent improves its return estimates over time thereby improving its skills. Specifically, given a novel task (i.e., a novel area of the state space), the agent's return estimates are likely to be highly inaccurate. When the agent deems its return estimates to be inaccurate, it will spend substantial time looking ahead and can exceed the time step duration leading to a low degree of flow ("the task is too hard"). As the agent learns the task, the return estimates become more accurate, the amount of deliberation approaches the duration of the time step, and the flow is maximized ("the task complexity matches the skills"). As the learning continues, the return estimates become even more accurate, requiring less lookahead time, and thus the degree of flow decreases ("the task is too easy"). When a task is mastered (i.e., learned) by the agent, the resulting amount of required lookahead corresponds to the *post-learning cognitive complexity of the task*. If this complexity is low, then the degree of flow the agent experiences is low, and the agent can switch to a new task to increase it.[1]

Effects of Maximizing Flow. Suppose the environment consists of a hierarchy of tasks (i.e., areas of the space) of increasing post-learning complexity. Then a flow-maximizing agent will try to master a task whose post-learning complexity requires just enough lookahead to match the time step duration, thereby maximizing the post-learning amount of flow. If it also happens that mastering tasks with high post-learning complexity carries a survival advantage, then maximizing flow is a potential evolutionary adaptation insomuch as agents that sense flow and seek to maximize it will tend to have higher fitness than agents that do not do so.

3.2 A Mathematical Model of Flow

To formalize our model, we adopt the standard Markov decision process model with a reinforcement learning agent operating in it (Sutton & Barto, 1998) and add a real-time constraint to it. We then extend the framework with the concept of flow.

1. "Civilization advances by extending the number of important operations which we can perform without thinking about them." (Whitehead, 1911)

3.2.1 The Reinforcement Learning Framework

Definition 1 Let S be the set of the agent's *states*. At each discrete time moment $t \in \{1, \dots, T\}$, the agent is in the state $s_t \in S$. It observes the state and computes its response – the *action* $a_t \in A$ where A is the set of all actions available to the agent. The environment then computes the next state s_{t+1} by stochastically drawing a state from the *transition probability* distribution $(p(s_t, a_t, s') \mid s' \in S)$. Here, for any two states $s, s' \in S$ and any action $a \in A$, $p(s, a, s')$ is the probability that the agent will end up in state s' by taking action a in state s. We denote the state draw operation as $s_{t+1} = \Lambda(s_t, a_t)$. By taking action a_t in state s_t, the agent collects a *reward* $r(a_t, s_t) \in \mathbb{R}$. The agent's objective is to select actions in such a way as to maximize the *total cumulative reward* $\sum_{t=1}^{\infty} \gamma^{t-1} r(a_t, s_t)$ where $\gamma \in (0, 1]$ is the *discount factor*.[2] The *return* $Q(s, a)$ of state s and action a is the expected cumulative discounted reward the agent can collect by starting in state s, taking action a and then following its policy. The returns are also called the *value function*.

Note that, while rewards represent immediate benefits that the agent accrues by taking an action in a state, returns represent long-term benefits. To illustrate, the action of undergoing a surgery can have a low reward (e.g., the post-surgical pain) but lead to a state with a high return (e.g., healing the agent). Thus, the agent should act so as to get to states with high returns.

Definition 2 The maximum possible return of the state s and the action a for any agent is $Q^*(s, a)$.

If the agent had access to Q^*, then it could act reflexively (and yet optimally!) by always taking the action with the highest Q^*: $a_t^* = \arg\max_{a \in A} Q^*(s_t, a)$. The problem lies with the fact that optimal returns of various states/actions are neither given to the agent *a priori* nor directly perceived by it while acting. Thus, the agent must learn such returns from the rewards it is collecting.

Definition 3 At time t, the agent's approximation to (or estimate of) the return $Q^*(s, a)$ is denoted by $Q_t(s, a)$. Our reward-seeking agents uses a real-time version of value iteration to learn Q values. At each time step t, the agent acts ϵ_t-greedily by selecting a random action ϵ_t percent of the time and acting greedily otherwise. The greedy action selection takes the action with the highest expected return, as predicted by its current value function $a_t \leftarrow \arg\max_{a \in A} Q_t(s_t, a)$. We compactly denote the ϵ-greedy action selection by $a_t \leftarrow \arg\max^{\epsilon_t}_{a \in A} Q_t(s_t, a)$.

The ϵ-greedy action selection is done to balance exploration of the environment and exploitation of the return estimates learned by the agent so far. The exploration parameter ϵ_t starts high at $t = 1$ and is gradually "cooled" toward zero as t approaches the agent's life time T.

3.2.2 Enhancing Return Estimates with Lookahead

The action-selection rule above relies on Q_t — estimates of the true state-action values. With inaccurate Q_t values looking ahead can improve action selection, similarly to game-playing programs and real-time heuristic search (Korf, 1990). We use the form of look ahead summarized in Figure 2 and Algorithm 1.

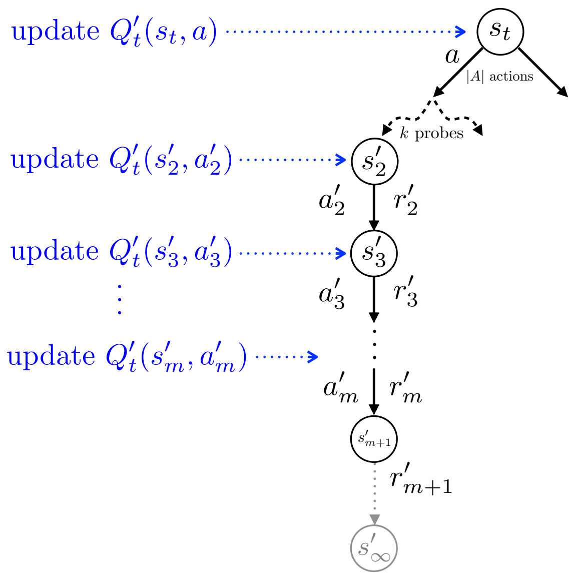


Figure 2. For each candidate action a, the agent launches k hypothetical probes of m steps each (a single probe is shown). The rewards imagined along such probes are used to enhance the Q values of the state-action pairs along each probe.

In a given state s_t, the agent determines the per-action amount of deliberation n_t in line 3, Algorithm 1. The agent factors this amount into k_t and m_t and then for each action (line 7) runs k_t probes (line 8) of m_t actions each (line 11). A probe to evaluate action $a \in A$ is an imaginary sequence of actions applied from the current state on. It is used to imagine future rewards the agent is likely to collect by starting in the current state s_t with the action a and then acting according to its return estimates. The rewards imagined during lookahead are used to enhance the return estimates for the current time step. The enhanced estimates are denoted by Q'_t in the pseudo-code.

Each probe starts in the agent's current state $s_t = s'_1$ (line 9) and action $a = a'_1$. The agent then imagines the possible resulting state s'_2 and the reward it would collect $r'_1 = r(s'_1, a'_1)$. To compute the state resulting from application of the action a in the state s, the agent draws a state s' from S according to the distribution $(p(s, a, s') \mid s' \in S)$. The agent then selects the next action according to its return estimates: $a'_2 = \arg\max_{a \in A} Q'_t(s'_2, a)$ (line 12) and the cycle repeats for m_t actions (line 11). As a result of the probe, the agent will have a sequence of imagined states: $(s'_1 = s_t, s'_2, \ldots, s'_{m_t+1})$, a sequence of actions taken between these states: $(a'_1 = a, a'_2, \ldots, a_{m_t})$ and the sequence of rewards that would be collected when taking the actions: $(r'_1, r'_2, \ldots, r'_{m_t})$. After the probe terminates in the state s'_{m_t+1} the agent estimates the cumulative reward it would be able to collect from that state on, if the probe were allowed to run indefinitely: $r'_{m_t+1} = \max_{a \in A} Q'_t(s'_{m_t+1}, a)$ (line 16).

2. Note that there is a disconnect between evaluating the agents which try to maximize an infinite discounted sum of rewards by running them a finite number of steps. Such artificial reward discounting is a common practice in the field of RL, as doing otherwise requires including the remaining life time $(t_{\max} - t)$ in the state description.

Algorithm 1: Algorithm for real-time Q-learning with lookahead (γ, r, p, T)

```
1  initialize $Q_1$ to uniformly random values in $[0, 1]$
2  for $t = 1, 2, \ldots, T$ do
3      determine the amount of deliberation: $n_t \leftarrow \Delta(s_t)$
4      set probe length: $m_t \leftarrow \lfloor\sqrt{n_t}\rfloor$
5      number of probes: $k_t \leftarrow \lfloor\sqrt{n_t}\rfloor$
6      initialize enhanced state-action return estimates: $Q'_t \leftarrow Q_t$
7      for $a \in A$ do
8          for $i = 1, \ldots, k_t$ do
9              $s'_1 \leftarrow s_t$
10             $\gamma_1 \leftarrow 1$
11             for $j = 1, \ldots, m_t$ do
12                 select probe action: $a'_j \leftarrow \begin{cases} a & \text{if } j = 1 \\ \arg\max_{a' \in A} Q'_t(s'_j, a') & \text{otherwise.} \end{cases}$
13                 imagine the next state: $s'_{j+1} \leftarrow \Lambda(s'_j, a'_j)$
14                 imagine the reward to be collected: $r'_j \leftarrow r(s'_j, a'_j)$
15                 adjust the discount factor $\gamma_{j+1} \leftarrow \gamma_j \gamma$
16             imagine the residual reward $r'_{m_t+1} \leftarrow \max_{a \in A} Q'_t(s'_{m_t+1}, a)$
17             for $j = 1, \ldots, m_t$ do
18                 update $Q'_t(s'_j, a'_j) \leftarrow (1 - \alpha_t) Q'_t(s'_j, a'_j) + \alpha_t \sum_{x=j}^{m_t+1} \gamma_x r'_x$
19                 adjust discount factor: $\gamma_x \leftarrow \gamma_x / \gamma, x \in \{j + 1, \ldots, m_t + 1\}$
20     execute the action: $a_t \leftarrow \arg\max^{\epsilon_t}_{a \in A} Q'_t(s_t, a)$
21     observe the resulting state $s_{t+1}$
22     update the return estimate using the collected reward and the new state:
       $Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha_t) Q_t(s_t, a_t) + \alpha_t [r(s_t, a_t) + \gamma \max_{a \in A} Q_t(s_{t+1}, a)]$
```

The agent then updates its $Q'_t(s'_j, a'_j)$ values along the probe ($j \in \{1, \ldots, m_t\}$) in line 17. Each $Q'_t(s'_j, a'_j)$ is updated with respect to the reward values imagined for the rest of the probe: $r'_j, r'_{j+1}, \ldots, r'_{m_t+1}$ in line 18. The updates use a learning rate (step size) α_t which is "cooled" towards 0 over time. Our way of updating the value function from imaginary experience is similar to the Dyna architecture (Sutton, 1990).

Once k_t probes have been run for each action, the agent ϵ_t-greedily selects the action a_t in line 20 using the lookahead-enhanced return estimates Q'_t. As described earlier, the exploration parameter ϵ_t is "cooled" towards 0 over time (not shown in the pseudo-code). Finally, the agent updates its return estimate using the actual collected reward in line 22. We use the standard Bellman-update learning rule with the step size α_t. Note that the lookahead-enhanced return estimates (Q'_t) are used only to select the next action. Actual states, rewards and actions are used to update Q_t which are subsequently carried over to the next time step.

3.2.3 Dynamic Selection of Lookahead Depth

Note that the probes are not run in the environment but are simulated in the agent's mind. A larger number of longer probes is likely to give better estimates of each state-action value $Q'_t(s,a)$ and thus improve the agent's action selection. However, simulating longer probes takes more time and thus may be disadvantageous in a real-time environment. Specifically, if the total work of running k_t probes of m_t actions each exceeds the maximum planning time $t_{\max}$ allotted to each action (i.e., $m_t k_t > t_{\max}$) then the agent receives a punishment in the form of a negative reward $\phi_t < 0$.

The amount of deliberation depends on the current state s_t and is computed by the function Δ in line 3. Intuitively, the amount of deliberation should depend on the accuracy of the agent's return estimates $Q_t(s_t, \cdot)$. Specifically, the agent should run just enough probes so that the Q_t enhanced to Q'_t by the lookahead will allow it to select an optimal action:

$$Q^*(s_t, \arg\max^{\epsilon_t}_{a \in A} Q'_t(s_t,a)) = \max_{a \in A} Q^*(s_t,a).$$

Without knowing the true values Q^*, it may be impossible for the agent to determine the minimum amount of lookahead it needs in the current state. Indeed, selecting the right lookahead depth has been an open research question (Bulitko et al., 2008; Luštrek & Bulitko, 2006; Bulitko et al., 2003). The problem difficulty is further compounded when the return estimates are not tabular but use state features (e.g., when $Q(s,a)$ is a weighted sum of features of state s). In this paper, we assume that the lookahead depth selection function Δ is provided to the agent (e.g., evolved via simulated evolution).

3.2.4 Flow

Described thus far, our mathematical model is a real-time variant of the simple value iteration similar to real-time dynamic programming (Barto, Bradtke, & Singh, 1995). We assume the environment contains different subareas of the state space, called tasks, such that different tasks require the agent to use different lookahead depths in order to achieve optimal action selection. In other words, different tasks have different post-learning complexity. If more complex tasks carry a higher reward then a reward-maximizing agent should find and learn the task whose post-learning complexity matches the agent's cognitive ability. Technically, this means that the lookahead depth the agent needs to select optimal actions for the task (after learning) matches the amount of time available in the environment ($t_{\max}$). Indeed, the lookahead required for yet more complex tasks will lead to exceeding $t_{\max}$ time available per action, thereby lowering the overall performance (and the reward collected). In psychological terms, the task will be too hard and will result in frustration. On the other hand, mastering less complex tasks brings the lookahead under $t_{\max}$. However, less complex tasks also carry lower reward, which decreases the lifetime reward the agent collects. Psychologically speaking, the task is too easy, and the agent is bored.

It is possible to find the task with post-learning complexity matching the agent's cognitive ability merely by trial and error (i.e., without any explicit awareness of flow). However, every error the agent makes (e.g., taking on a task that is too complex or too simple) has a lost opportunity cost. By equipping our agents with an explicit awareness of flow, finding the task of a matching complexity can be accelerated, resulting in higher lifetime rewards.

Definition 4 The *degree of flow* $F(s)$ experienced by the agent in the state s is inversely proportional to over- or under-utilization of the agent's cognitive resources. Mathematically, $F(s) = f/(|n_t - t_{\max}| + \xi)$ where n_t is the per-action amount of deliberation the agent expended in state s. The maximum amount of deliberation allowed per action, $t_{\max}$, is a domain-specific constant and ξ is a small positive constant to keep $F(s)$ bounded when $n_t = t_{\max}$. The constant f is the *flow-awareness coefficient* and is specific to the agent.

We incorporate the flow into the agent's decision-making by introducing F into the agent action-selection rule (line 20 of Algorithm 1). The flow-enhanced version is:

$$a_t \leftarrow \arg\max{}^{\epsilon_t}_{a \in A} \left[Q'_t(s_t, a) + \sum_{s \in S} p(s_t, a, s) F(s) \right]$$

which means that the agent prefers to take actions that lead to states with a higher expected degree of flow. The values of F are initially random over all states but are updated by the agent as:

$$F(s_t) \leftarrow \frac{f}{|n_t - t_{\max}| + \xi}$$

at each time step. Note that the flow values of states, F, exist only in the agent's mind and are agent-specific since the flow-awareness coefficient f is an agent-specific parameter. If they were a part of the environment, they would make it non-Markovian since each $F(s_t)$ would depend not only on the agent's current state s_t but also on the amount of deliberation n_t expended by the agent in the state. Thus, we keep F separate from the actual rewards received from the environment.

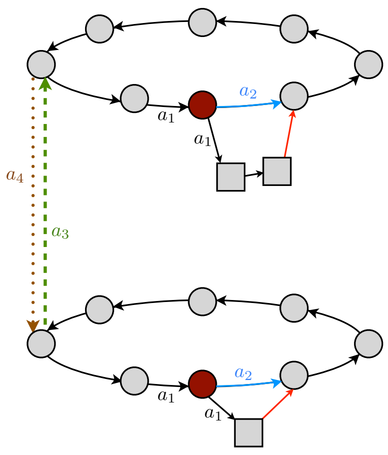
4. Empirical Evaluation

Given the definition of flow in the reinforcement learning framework presented above, a natural question is whether maximizing flow can give the agent a boost in performance — a higher lifetime reward. In this section, we present a simple study which gives an empirical answer to the question above. We start by developing a simple testbed, then describe an implementation of the agent framework in it and finally show the results.

4.1 The Stacked Rings Environment

We need an environment containing tasks of various cognitive complexity. In the MDP setting, the tasks are subgraphs of the state transition graph. The complexity will come from the amount of lookahead the agent needs to master each subgraph. In the following, we satisfy these requirements with a simple and scalable deterministic MDP.

The deterministic transition graph is a hierarchy of *rings*. Visually, we can think of a stack of I rings (Figure 3). Each rings has N main states, where N is a multiple of 4. The action a_1 moves the agent along the current ring. Each application of a_1 gives the agent a reward of i/I which means that higher rings are more rewarding. In the example in the figure, the agent can collect the per-action reward of $1/2$ by going around the bottom ring and the per-action reward of $2/2$ at the



action	next state	reward
a_1	next main ring state *or* side loop	i/I
a_2	same state *or* next main ring state	-1 *or* 1
a_3	same state *or* one ring up	-1
a_4	same state *or* one ring down	-1
a_1	next side loop state *or* main ring	i/I *or* $-2i(i+3)/I$
a_2	same state	-1
a_3	same state	-1
a_4	same state	-1

Figure 3. **Left:** a simplified visualization of the stacked ring environment for two main rings ($I = 2$) of eight states each ($N = 8$). **Right:** transition and reward models for actions taken in main ring states (top half of the table); transition and reward models for actions taken in side loop states (bottom half of the table).

top ring. As also shown in the figure, the agent can switch rings by taking actions a_3 (going up one ring) and a_4 (going down one ring) when it is in certain states on each ring. Every fourth main ring node has connections to the adjacent upper and lower rings allowing the agent to go up (a_3) or down (a_4). Otherwise, actions a_3 and a_4 simply return the agent to the same state. These actions are shown as dotted/dashed arrows in Figure 3.

As described so far, a reward-maximizing agent ought to travel to the highest ring and then circle around it forever. This is due to all rings' having identical cognitive complexity, based on the description so far. To make higher rings more complex, we introduce *traps*. These are implemented as *side loops* attached to the main rings every four states. For instance, with eight main states on each ring, two side loops are attached to each ring (only one side loop per ring is shown in the figure for the sake of clarity). Side loop states are shown as squares whereas main ring states are shown as circles in the figure. The side loops function as traps because once the agent enters a side loop, it has to follow it through and will get hit with a large negative reward at the end of the loop as it returns onto the main ring. The punishing action in each side loop is shown in red in the figure. The length of a side loop is equal to the ring number i to which it is attached. So, side loops attached to ring 1 are 1 state long. Side loops attached to ring two are two states long, and so on.

In order for the agent not to fall into a side loop, it needs to know that the usually safe action a_1 (black arrows in the figure) leads into a side loop in certain states (shown by darker red circles in the figure). In such *trap-entry states* the agent needs to take the action a_2 instead (light blue in the figure) to remain on the main loop. Note that the action a_2 is usually a bad action to take since in all other main ring states it keeps the agent in the same state but penalizes it with the reward of -1.

If the agent could represent returns for each state accurately and independently then it would learn to take the action a_2 at the trap-entry states and the action a_1 otherwise. As a result, it would always stay on a main ring and collect maximum reward. Suppose, however, that the agent's cognitive ability is limited and it can represent return estimates only at the level of rings and not at the level of individual states on a ring. That means that it cannot distinguish (return-wise) between a normal main ring state and a trap-entry state. This can happen, for instance, if return estimates $Q(s, a)$ are represented in a non-tabular way (e.g., as a linear combination of some features of the state s and the action a).

Our agents compensate for inaccuracies in their return estimates via lookahead. If the agent is in a trap entry state s, facing the danger of falling into a side loop of i states then it needs to run a lookahead of at least depth i so that the lookahead-enhanced return $Q'(s, a_1)$ is different from the lookahead-enhanced return $Q'(s, a_2)$. In our environment the length of each side loop attached to ring i is i states. That means that side loops attached to more rewarding rings require deeper lookahead to avoid. In summary, each ring represents a task. Higher rings are more rewarding when mastered but require a deeper lookahead to avoid the large punishment at the end of each side loop. Thus, higher rings, viewed as tasks to master, have a higher cognitive complexity.

4.2 Hypotheses

With the cognitive limitations described in the previous section, the agent must set the probe length as $m_t = i$ where i is the ring number for the current state s_t to avoid falling into side loops. Consequently, the amount of lookahead will be $\Delta(s_t) = n_t = i^2$. Then the optimal course of action is to move to the state whose ring number is $i^* = \lfloor\sqrt{t_{\max}}\rfloor$ and stay in that ring. Intuitively, i^* is the ring of the appropriate complexity, given the agent's inability to distinguish between states along a ring in its representation of return estimates. Indeed, rings above i^* are too complex for the agent insomuch as the lookahead required to enhance Q_t enough to avoid falling into the side loops would exceed $t_{\max}$, causing the agent to incur the punishment ϕ_t. Conversely, rings below i^* can indeed be mastered by the agent but give less reward than the ring i^*.

On the other hand, if the agent does use a tabular representation of Q_t and an ϵ-greedy action-selection rule, we expect it to frequently[3] converge to the optimal ring i^* even in its basic version (i.e., without any flow awareness). This is **Hypothesis 1**.

Note that main states on the optimal ring i^* have the highest degree of flow (up to f/ξ). Subsequently, a flow-maximizing agent will be drawn to the states on the optimal ring. As a result, the agent will spend more time on the optimal ring i^* and thus collect a higher lifetime reward. Higher values of the flow-awareness coefficient f should then increase the overall reward collected (**Hypothesis 2**). Viewed from another angle, higher values of the flow-awareness coefficient reduce the number of steps needed to reliably reach the optimal ring (**Hypothesis 3a**). Or, given a fixed number of learning steps, higher values of the flow-awareness coefficient will cause the agent to reach the optimal ring more frequently (**Hypothesis 3b**).

4.3 Results

We ran the agent in an environment consisting of $I = 10$ stacked rings, each with $N = 8$ main ring states along with side loops as described in Section 4.1. The total number of states on the main rings and the side loops was 190. The action, transition and reward structure was set as per Section 4.1.

3. The ϵ-greedy learning process is stochastic and convergence to i^* is not guaranteed.

The agent was run with the values $\{0, 200, 400, 600, 800, 1000\}$ for the flow-awareness coefficient f. For each value of f we ran the agent for $10,000$ steps and measured the total non-discounted non-flow reward it collected from the environment. The discount factor used by the agent to learn its return estimates was fixed at 0.99. The learning rate was "cooled" as $\alpha_t = 0.9/\sqrt[10]{t}$. The exploration rate started at 0.8 and was linearly brought to zero in approximately half the life time of the agent. It was 0 for the second half of the agent's life. Since the learning process is stochastic, we repeated each experiment 3600 times. The constant ξ used to compute F was set to 1. The agent started in a main ring state on the lowest ring. The start state was not a trap entry.

We repeated each of the 3600 trials for three different values of the maximum amount of deliberation per action, $t_{\max} \in \{1.5, 25.5, 100.5\}$. The agent's amount of deliberation in state s was set to i^2 where i is the ring number of the state s: $n_t = \Delta(s_t) = i^2$. If, at time step t, the agent's deliberation amount n_t exceeded $t_{\max}$, it received a negative reward of $\phi_t = -0.25\lfloor n_t/t_{\max}\rfloor$.

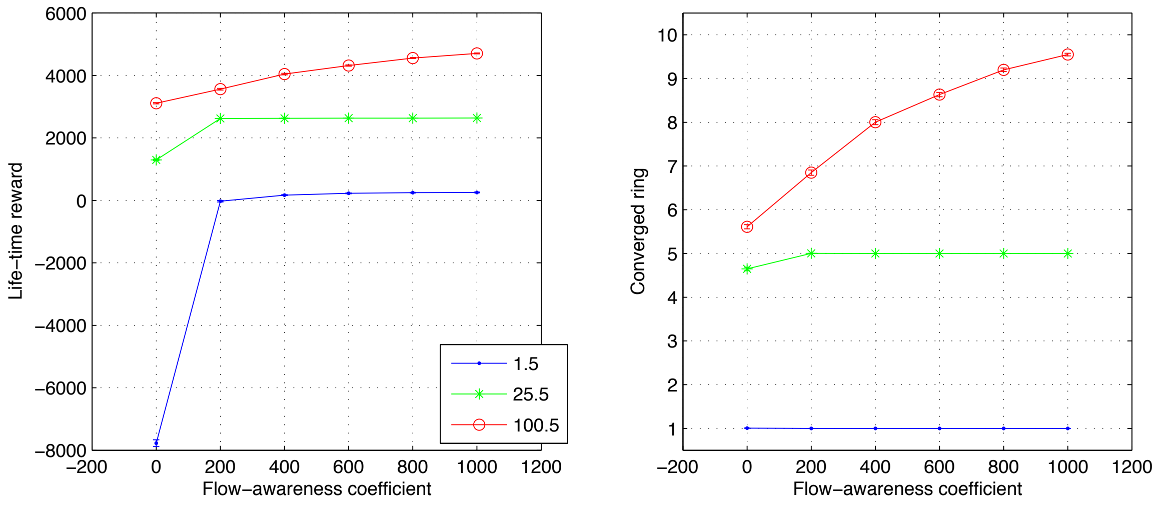


Figure 4. Agent performance for different values of the flow-awareness coefficient f and the per-action deliberation limit $t_{\max}$. The error bars show the standard error of the mean. Different lines correspond to different values of $t_{\max}$ as shown in the legend. The agent life time was $10,000$ steps and we ran 3600 trials.

The results of 3600 trials for each combination of the flow-awareness coefficient f and the maximum amount of per-action deliberation $t_{\max}$ are shown in Figure 4. Each point is an average over 3600 trials. The error bars show the standard error of the mean. As the graph on the left shows, higher values of the flow-awareness coefficient allowed the agent to collect higher lifetime reward. One-way ANOVAs revealed that for $t_{\max} = 1.5$, there was a significant main effect of the flow-awareness coefficient f: $p < \omega$ where ω is the smallest positive floating-point number MATLAB can represent, $F = 4680$, $df = 5, 21594$. There was also a significant effect of f for $t_{\max} = 25.5$ ($p < \omega$, $F = 2509$, $df = 5, 21594$) and for $t_{\max} = 100.5$ ($p < \omega$, $F = 840$, $df = 5, 21594$). The data thus supports **Hypothesis 2**.

We also analyzed the ring number the agent converged to at the end of its life time, averaged over 3600 trials.[4] These values are plotted in the right graph of Figure 4. For $t_{\max} = 1.5$, the agent always converged to ring 1. For $t_{\max} = 25.5$, flow-awareness coefficient had a significant effect on converged ring (one-way ANOVA, $p < \omega$, $F = 74$, $df = 5, 21594$). For $t_{\max} = 100.5$, flow-awareness coefficient also had a significant effect on converged ring (one-way ANOVA, $p < \omega$, $F = 945$, $df = 5, 21594$). The data support **Hypothesis 3b**.

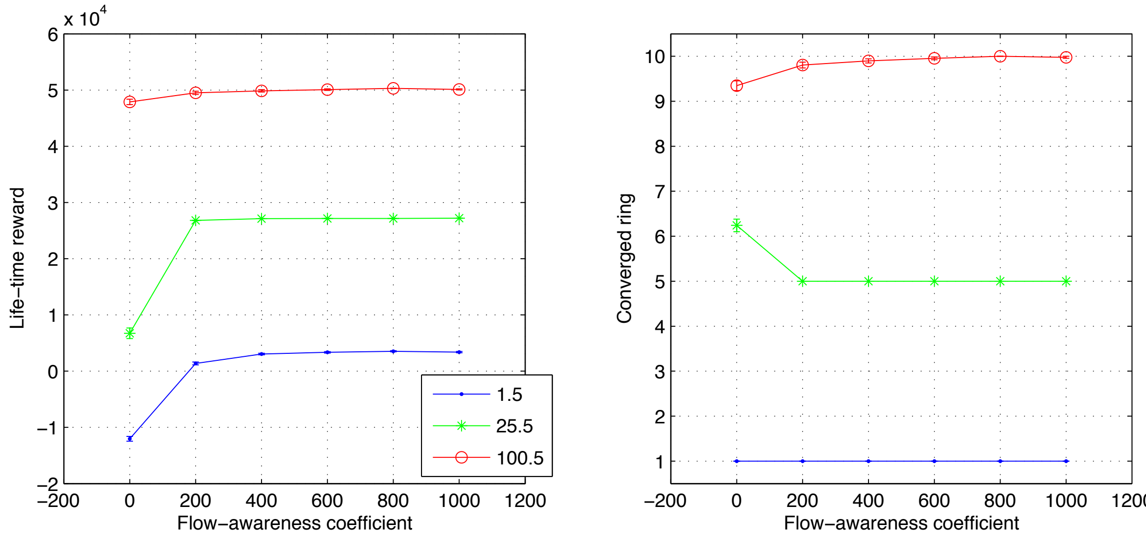


Figure 5. Agent performance for different values of the flow-awareness coefficient f and the per-action deliberation limit $t_{\max}$. The error bars show the standard error of the mean. Different lines correspond to different values of $t_{\max}$ as shown in the legend. The agent life time was $100,000$ steps and we ran 360 trials.

Both hypotheses continue to be supported by empirical data when we increase the number of learning steps from 10 to $100,000$ (Figure 5). One-way ANOVAs revealed that for $t_{\max} = 1.5$, there was a significant main effect of the flow-awareness coefficient f on lifetime reward ($p < \omega$, $F = 763$, $df = 5, 2154$). There was also a significant effect of f for $t_{\max} = 25.5$ ($p < \omega$, $F = 472$, $df = 5, 2154$) and for $t_{\max} = 100.5$ ($p < 10^{-11}$, $F = 12.26$, $df = 5, 2154$). For $t_{\max} = 1.5$, the agent always converged to ring 1. One-way ANOVAs revealed a significant main effect of the flow-awareness coefficient f on the converged ring for $t_{\max} = 25.5$ ($p < 10^{-74}$, $F = 78$, $df = 5, 2154$) and for $t_{\max} = 100.5$ ($p < 10^{-17}$, $F = 19$, $df = 5, 2154$).

To investigate **Hypothesis 3a** we designed the following experiment. Instead of setting the agent's life time *a priori*, we started with only 200 steps. We then checked if the 200 steps are enough to reach the optimal ring i^* in each of 50 consecutive trials. If in one of the trials the agent failed to converge to the optimal ring at the end of its life time, we increased the life time of the agent by 40% and ran 50 more trials of $200 \times 1.4 = 280$ steps each. This process of incrementally increasing the agent's life time was carried out until either the agent was able to reach the optimal ring on each of 50 consecutive trials or a lifetime limit of $250,000$ steps was reached.

4. The agent is said to converge to ring x if x is the ring number on the very last time step of agent's life.

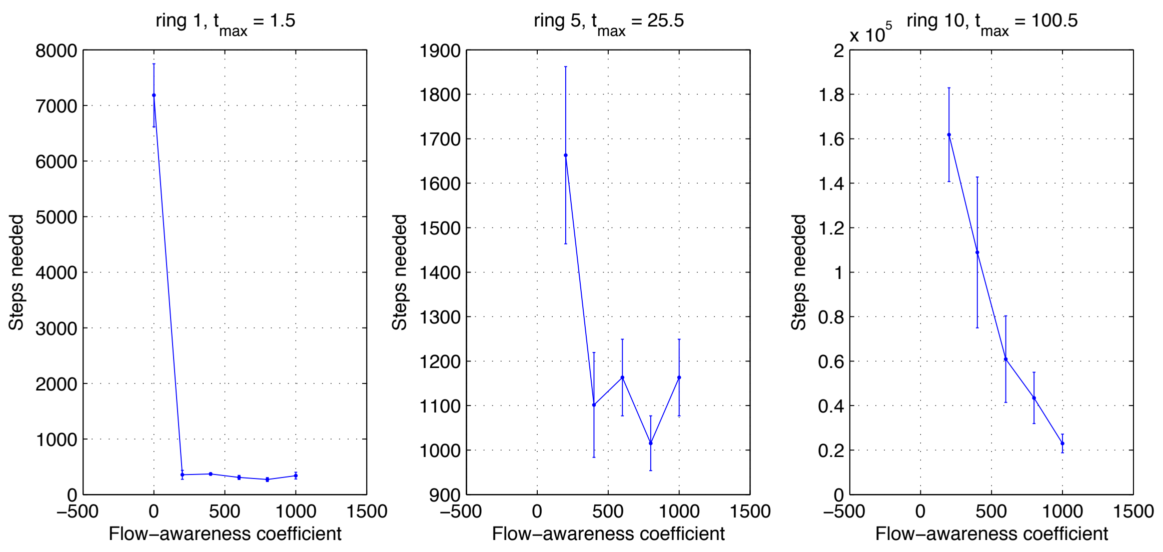


Figure 6. Lifetime steps needed to converge to the optimal ring for 50 trials in a row. Missing plot points indicate exceeding the limit of $250,000$ steps.

We repeated the entire process 18 times for six values of the flow-awareness coefficient $f \in \{0, 200, 400, 600, 800, 1000\}$ and three values of $t_{\max} \in \{1.5, 25.5, 100.5\}$ which induced the optimal ring $i^* \in \{1, 5, 10\}$ respectively. We then averaged the results over five independent runs. The means and the standard errors of the mean are plotted in Figure 6 where missing data points represent exceeding the limit of $250,000$ lifetime steps on at least one of the five runs. Higher values of the flow-awareness coefficient appear to reduce the number of steps necessary to reliably reach the optimal ring thereby supporting **Hypothesis 3a**.

Looking at the right side of Figure 4, the agent without flow awareness ($f = 0$) does not appear to converge reliably to the optimal ring $i^* = 10$ for $t_{\max} = 100.5$. Indeed, the mean value of the converged ring over the 3600 trials is approximately 5.6 ± 0.05. We trust it is due to the insufficient amount of exploration over the short life time of the agent. Increasing the agent's life time by an order of magnitude (i.e., from $10,000$ to $100,000$ steps) improves the mean value of the converged ring to approximately 9.4 ± 0.1 as seen in Figure 5.[5] This supports **Hypothesis 1**.

4.4 Discussion

All three hypotheses are supported by the experimental data. This is expected as, mathematically, the punishment for exceeding the maximum amount of deliberation per action was set as $\phi = -0.25\lfloor n_t/t_{\max}\rfloor$. Since our agent's amount of deliberation per action was set as $n_t = i^2$ where i is the agent's ring number, the negative reward becomes $\phi = -0.25\lfloor i^2/t_{\max}\rfloor$. The negative reward is thus least punishing when $i = i^* = \lfloor\sqrt{t_{\max}}\rfloor$. Hence, effectively, the returns of states on rings different from the optimal ring i^* are diminished. The flow of state on ring i is computed as

5. The standard error of the mean is higher due to the smaller number of trials: 360 instead of 3600.

$F = f/(|i^2 - t_{\max}| + 1)$ which is maximized when $i = i^*$. Thus, flow drives the agent towards states on the optimal ring, and the drive is stronger for higher values of the flow-awareness coefficient f.

To illustrate, if $t_{\max} = 25.5$, the optimal ring $i^* = \lfloor\sqrt{25.5}\rfloor = 5$. Avoiding the side loops, the per-step reward on that ring is $5/10 = 0.5$ for the action a_1 and 1 for the action a_2. Without a time limit, ring 6 would yield a higher per-step reward of $6/10 = 0.6$ (a_1) or 1 (a_2). However, the punishment for being on ring 6 is $\phi = -0.25\lfloor 6^2/25.5\rfloor = -0.25$, which brings the best possible per-step reward to $0.6 - 0.25 = 0.35$ or $1 - 0.25 = 0.75$. This makes staying on ring 6 less rewarding than staying on ring 5. Thus, even without the flow the agent would prefer ring 5 to ring 6. With the flow-awareness coefficient $f = 100$, the degree of flow for states on ring 5 is $F = 100/(|25 - 25.5| + 1) \approx 66.7$. The degree of flow for states on ring 6 is $F = 100/(|36 - 25.5| + 1) \approx 8.7$. Thus, the flow further encourages the agent to stay on ring 5 as opposed to moving up to ring 6.

This analysis illustrates how flow-seeking agents are drawn to master the tasks with post-learning complexity that matches their cognitive abilities. Mathematically, the additional flow rewards re-shape the reward stream of the environment perceived by the agent, making it easier for a reward-maximizing agent to converge to the optimal ring.

Connecting the results to human cognition, we speculate that the ability of our flow-aware agents to collect higher lifetime reward by more quickly and reliably converging to the task of matching complexity may be paralleled in people. Specifically, individuals with a highly developed sense of flow may behave similarly to flow-maximizing agents with a higher flow-awareness coefficient. That is, they are able to identify their “calling” more reliably and faster and, therefore, enjoy their life more. Similar phenomena may take place at shorter temporal scales (e.g., finding the hobby of a matching complexity or even locating a matchingly stressful route for one’s commute to work). We would like to investigate this hypothesis via a user study as a part of future work.

5. Generality of the Model

The flow model presented in this paper is not limited to reinforcement learning. Indeed, the key idea lies with giving an agent an ability to sense and a desire to maximize the degree of flow. Doing so will guide the agent to select a task of the “right” cognitive complexity.

We believe this meta-cognition mechanism is valuable in two ways. First, it may give an insight into human/animal selection of cognitive tasks. It may well be that evolution equipped humans with a “flow meter” (e.g., in the form of happiness) and a desire to maximize the perceived flow. Sensing and maximizing the flow would be an evolutionary adaptation as it would allow humans/animals to select the most appropriate tasks faster and more reliably. Second, our model should be applicable to a wide variety of cognitive architectures, including those that reason over structured symbolic representations (as opposed to the flat numeric model of the basic value iteration).

To illustrate, consider PRODIGY — a domain-independent search-based planner that uses different learning modules to become more efficient in its search (Carbonell et al., 1991). Given a domain description expressed in predicate logic, the cognitive skill of PRODIGY is related to the collective power of search control rules it has previously learned. As PRODIGY solves more problems, it automatically extends its control knowledge. The cognitive complexity of a planning problem is related to the collective power of control rules that would be *needed* to restrict the search tree enough to make the search tractable. PRODIGY's control rules learned from solving one problem can be then used to help solve another problem. So it may be a good strategy to take on less complex planning problems first and derive additional control rules in the process of solving them.

Our model of flow can guide PRODIGY through a hierarchy of problems of increasing complexity as follows. Given a planning problem to solve (i.e., the domain theory, the initial state and the goal expression), we can define the degree of flow as the match between the size of the maximum tractable search tree that contains a solution and the size of the search tree that PRODIGY would expand with its current body of control rules. The closer the match, the higher the amount of flow it would experience while solving the problem. Having equipped PRODIGY with such a "flow meter" and a desire to maximize flow, we would then give the system a choice to take on any planning task from a large body of problems of various complexities. We would expect PRODIGY to progress through problems of increasing cognitive complexity, efficiently building up its control rule base in the process. Its flow meter will effectively serve as a meta-cognition mechanism, determining its choice of the problems.

6. Current Shortcomings and Future Work

In this preliminary study we made a number of simplifying assumptions. First and foremost, we used tabular learning for the return estimates while forcing the amount of lookahead that would be necessary for an agent estimating returns on a per-ring basis. We believe this approximates the more realistic case of the agent using a function approximation for its return estimates, based on features of states. Future work will test this belief by actually using function approximation to learn returns as well as considering a more flexible schema for lookahead selection.

Second, we used a single simple, small and deterministic environment where the punishment for missing a real-time deadline was in the form of a negative reward. While we feel this approach shares some important properties of large-scale real-world environments, future work will test it empirically, using varied penalties for missing a deliberation deadline. This will be done in the context of more realistic and complex environments. One possibility is to use a competitive real-time environment such as the one used for the international *Ms. Pac-Man versus Ghosts* competition (Rohlfshagen, 2012). We speculate that the *Pac-Man* environment contains a hierarchy of tasks of different complexity and rewards. For instance, the task of merely collecting the pellets is simple and somewhat rewarding. The task of collecting the pellets while staying away from ghosts is more complex and more rewarding. The task of seeking the power pellets and subsequently hunting then-vulnerable ghosts may be more complex and rewarding still.

Third, if an agent takes on a novel complex task, its lookahead may need to be high, leading to low degrees of flow. This may "discourage" the agent and drive it back to the task it has already mastered. We believe this can be mitigated with cross-task portability of skills in real life, making learning a more complex task easier once the agent masters a relevant simpler task. In our framework, such skill portability will reduce the required lookahead on a novel task by re-using some of the learned knowledge for a simpler task. This appears applicable to the task hierarchy of *Pac-Man* where more complex and rewarding tasks may include easier tasks as subtasks.

Fourth, we connected the cognitive complexity of tasks to the cognitive skill level of an agent via the amount of planning time the agent needs to spend per action in order to handle the task. Other cognitive resources need to be considered. For instance, the degree to which an agent is able to map low-level sensory inputs to higher-level abstract symbols, as well as reason in the space of such abstract representations, is a cognitive skill. The skill can determine which tasks the agents can take on successfully as well as the degree of flow the agent experiences when it takes on tasks with a certain amount of abstract reasoning required. The amount of short and long-term memory available to the agent is another measure of the agent's cognitive skill level which can be used for matching tasks to the agent's skills as well as determining the degree of flow.

Finally, the flow-awareness coefficient, the exploration coefficient schedule and the learning step size are agent-specific. While in this study we chose them manually via trial and error, future work will investigate the extent to which such parameters can be selected automatically. We plan to encode the parameters in the agent's genome in simulated evolution and then show that flow-maximizing agents have greater evolutionary fitness. We further speculate that maximization of flow can emerge as an evolutionary adaptation in a framework similar to evolutionary reinforcement learning (Ackley & Littman, 1991).

7. Conclusions

In this paper, we presented a simple model of a psychological condition called "flow". We proposed that the degree of flow is defined by how well the agent's cognitive skills match the cognitive complexity of the task: the better the match, the higher the degree of flow the agent experiences. If the environment contains a hierarchy of tasks of increasing cognitive complexity and reward, then the agent can optimize its performance by taking on a task of matching cognitive complexity. This will also increase the degree of flow it perceives. The converse also holds: flow-maximizing agents will take on tasks of matching complexity for the sake of flow, incidentally improving their performance.

Exploiting this bi-directional connection, we equipped our agents with an explicit sense of the degree of flow they experience, making them flow-aware. Such flow-aware agents can excel over their non-flow aware counterparts in their cognitive performance, by choosing the right task. We implemented these ideas in the standard reinforcement learning framework by expressing the flow awareness as an additional stream of rewards. We then showed that agents equipped with such a reward stream collect more of the environmental (non-flow) reward.

Acknowledgements

We appreciate the support of the National Science and Engineering Research Council. We are grateful for discussions with the members of the Making Minds Reading Group at the University of Alberta, Valeriy and Ina Bulitko, Pat Langley and Thórey Maríusdóttir. We also appreciate feedback from the anonymous reviewers.

References

Ackley, D. H., & Littman, M. L. (1991). Interactions between learning and evolution. In C. Langton, C. Taylor, J. D. Farmer, & S. Ramussen (Eds.), *Artificial Life II*, Vol. 10, 487–509. Redwood City, CA: Addison-Wesley.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*, 81–138.

Bulitko, V., Li, L., Greiner, R., & Levner, I. (2003). Lookahead pathologies for single agent search. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (pp. 1531–1533). Acapulco, Mexico: Morgan Kaufmann.

Bulitko, V., Luštrek, M., Schaeffer, J., Björnsson, Y., & Sigmundarson, S. (2008). Dynamic control in real-time heuristic search. *Journal of Artificial Intelligence Research*, *32*, 419–452.

Carbonell, J., Etzioni, O., Gil, Y., Joseph, R., Knoblock, C., Minton, S., & Veloso, M. (1991). PRODIGY: An integrated architecture for planning and learning. *ACM SIGART Bulletin*, *2*, 51–55.

Csikszentmihalyi, M. (1990). *Flow: The psychology of optimal experience*. New York: Harper and Row. First edition.

Jin, S.-A. A. (2012). Toward integrative models of flow: Effects of performance, skill, challenge, playfulness, and presence on flow in video games. *Journal of Broadcasting & Electronic Media*, *56*, 169–186.

Korf, R. (1990). Real-time heuristic search. *Artificial Intelligence*, *42*, 189–211.

Lim, S. H., & Auer, P. (2012). Autonomous exploration for navigating in MDPs. *Proceedings of the Journal of Machine Learning Research 25th Annual Conference on Learning Theory* (pp. 40.1–40.24). Edinburgh, Scotland.

Luštrek, M., & Bulitko, V. (2006). Lookahead pathology in real-time path-finding. *Proceedings of the National Conference on Artificial Intelligence, Workshop on Learning For Search* (pp. 108–114). Boston, MA: AAAI Press.

Rohlfshagen, P. (2012). The Ms. Pac-man versus Ghosts Competition. Retrieved November 27, 2012, from http://www.pacman-vs-ghosts.net.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224). Austin, TX: Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Weber, R., Tamborini, R., Westcott-Baker, A., & Kantor, B. (2009). Theorizing flow and media enjoyment as cognitive synchronization of attentional and regard networks. *Communication Theory*, *19*, 397–422.

Whitehead, A. N. (1911). *An introduction to mathematics*. London: Williams & Northgate.