Proceedings, The Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-15)

Maximizing Flow as a Metacontrol in Angband

Thorey Mariusdottir and **Vadim Bulitko**
Dept. of Computing Science, University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
mariusdo@ualberta.ca and bulitko@ualberta.ca

Matthew Brown
Dept. of Psychiatry, University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
mbrown2@ualberta.ca

Abstract

Flow is a psychological state that is reported to improve people's performance. Flow can emerge when the person's skills and the challenges of their activity match. This paper applies this concept to artificial intelligence agents. We equip a decision-making agent with a metacontrol policy that guides the agent to activities where the agent's skills match the activity difficulty. Consequently, we expect the agent's performance to improve. We implement and evaluate this approach in the role-playing game of *Angband*.

1 Introduction

Traditionally, decision making within artificial intelligence (AI) is framed as an action-perception loop: an AI agent observes the state of the environment and then uses its control policy to select and execute an action. The agent then perceives the new state resulting from the action, and the cycle repeats. One can also design AI agents that reason about their own decision making (Anderson and Oates 2007; Cox and Raja 2007). Such *metacontrol* (or metareasoning) is the monitoring and control of the decision-making cycle. It represents a higher layer of control that perceives and acts on the underlying decision-making process rather than just on the environment.

In this paper, we consider metacontrol based on the psychological concept of *flow*. Csikszentmihalyi (1975) defined flow as the highly enjoyable psychological state that humans experience when they are fully immersed in an activity. A key condition of the emergence of the flow state is a match between the person's skills and the difficulty of the activity. When the match is violated, boredom (skills exceed difficulty) or frustration (difficulty exceeds skills) sets in.

Not only is being in a state of flow intrinsically rewarding but the person's performance on the activity is also increased or even optimized (e.g., Carli, Fave, and Massimini 1988, Mayers 1978, Nakamura 1988). Thus, seeking flow is a form of metacontrol guiding people towards activities whose difficulty matches their skills, thereby optimizing their performance on the activity. In line with Bulitko and Brown (2012), we apply this metacontrol strategy to AI agents that can choose the activity they engage in.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

As an illustration, consider the video game *Angband* (Cutler et al. 1990). *Angband* is a dungeon exploration game divided into discrete levels. The player starts on level 0 and progresses into the dungeon with the objective of reaching the final level (100) and defeating a boss monster. Higher levels of the dungeon are more complex to negotiate and require higher player character skills. Each successful action (e.g., slaying a monster) gains the player experience points that can be used to level up, upgrading the skills of the player's in-game character. Players in *Angband* compete on their game score. Each slain monster adds to the score, but monsters whose in-game level is higher than the player's add more points to the score. Thus, in a given fixed amount of game time, progressing through the dungeon quickly and fighting difficult monsters can improve the player's score **if** the player manages to survive the difficult encounters. Score accumulation ends when the player's character dies.

In line with Bulitko and Brown (2012), we define flow as a continuous scalar variable (called the *degree of flow*) whose higher values indicate a better match between the agent's skills and the activity difficulty. We then conjecture that in certain environments, such as the game of *Angband*, seeking flow leads to improved performance. Indeed, in *Angband* the player needs to progress quickly to gain a higher score but not so quickly that they die before finishing the game. Thus, *Angband* appears to be well suited to a flow-maximizing metacontrol strategy. By choosing the dungeon level based on maximizing its flow, an AI agent will pace its descent through the dungeons, hopefully leading to a higher score.

This paper contributes the first empirical evaluation of a recent computational model of flow. The rest of the paper is organized as follows. Section 2 formally describes the problem. We then review related work on metacontrol policies in Section 3. Our flow-maximizing metacontrol is detailed in Section 4 and empirically evaluated in Section 5. We conclude the paper with directions for future work.

2 Problem Formulation

We model our problem as a stochastic decision-making process, defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{S}^\dagger, s_0 \rangle$. Here, $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions; $\mathcal{P}(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is a transition probability function that gives the probability of moving from state s to state s' by executing action a; $\mathcal{S}^\dagger$ is a set of absorbing terminal states

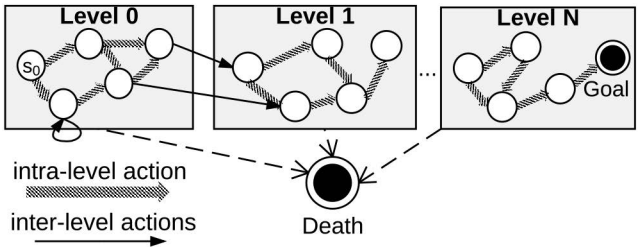

Figure 1: A state space partitioned into levels.

that is composed of a *goal state* s_g and *death state* s_d. The agent starts in the state s_0.

We consider state spaces partitioned into discrete levels (Figure 1). Following Bulitko (2014), we assume the state space, except the death state, is partitioned into a sequence of levels $\mathcal{L}_0, \ldots, \mathcal{L}_N$: $\mathcal{S} \setminus \{s_d\} = \bigcup_{i=0}^{N} \mathcal{L}_i$. The levels are non-empty, non-overlapping sets of states. The initial state $s_0 \in \mathcal{L}_0$, and the goal state $s_g \in \mathcal{L}_N$.

The set of all actions $\mathcal{A}$ is partitioned into two sets of actions: the set of *inter-level actions* A and the set of *intra-level actions* $\tilde{A}$. Inter-level actions intend to move the agent to a specific level. This can include the current level, but then it keeps the agent in its current state. Intra-level actions move the agent only within a level. The agent may reach the death state after any action.

The agent's policy is a mapping from the states to actions. Bulitko (2014) proposed that, for level-based environments, we decompose the policy of the agent into two parts. The *ground policy* $\tilde{\pi}$ is restricted to intra-level actions: $\tilde{\pi} : \mathcal{S} \to \tilde{A}$, and the *metacontrol policy* π is restricted to inter-level actions $\pi : \mathcal{S} \to A$. Generally speaking, ground and metacontrol policies need not share the same environment. For instance, the ground policy itself can be a part of the metacontrol's environment. In this paper, we assume the state space encompasses the full state of both the metacontrol and ground policy environments so that both policies map from $\mathcal{S}$ to their sets of actions.

The agent traverses the state space in the following fashion (Algorithm 1, adapted from (Bulitko 2014)). At a time step t, the agent performs two actions. First, the agent observes the current state s_t and performs an action a'_t selected by its metacontrol policy (line 3). The environment transitions to the new state s' (line 4). In that state, the agent performs an action a_t (line 5) selected by the ground policy. The environment transitions to the state s_{t+1} (line 6). The process continues until the agent either reaches the goal $s_g \in \mathcal{L}_N$ or transitions to the death state s_d (line 2).

In this paper, we only consider finding the best metacontrol policy (π) for a given fixed ground policy ($\tilde{\pi}$). We will evaluate such metacontrol policies using three performance measures. Given a metacontrol policy π, we will measure (1) the *expected time* that agents controlled by π take to reach the goal s_g, (2) the *failure rate*, that is, the probability of reaching the death state before reaching the goal, and (3) the *expected life-time reward*.

The third performance measure requires that the agent collect rewards during its lifetime. For example, in *Angband* the reward r_t received at time t can be defined as the experience points received for the actions a'_t and a_t.

Algorithm 1: Agent Operation

inputs: A stochastic decision-making process $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{S}^\dagger, s_0 \rangle$, ground policy $\tilde{\pi}$, and metacontrol policy π
output: trajectory $(s_0, s_1, \ldots, s_T)$, $s_T \in \mathcal{S}^\dagger$

1 initialize $t \leftarrow 0$, $s_t \leftarrow s_0$
2 **while** $s_t \notin \mathcal{S}^\dagger$ **do**
3 apply the metacontrol policy $a'_t \leftarrow \pi(s_t)$
4 observe the next state $s' \xleftarrow{\mathcal{P}(s'|s_t, a'_t)} s_t$
5 apply the ground policy $a_t \leftarrow \tilde{\pi}(s')$
6 observe the next state $s_{t+1} \xleftarrow{\mathcal{P}(s_{t+1}|s', a_t)} s'$
7 advance time $t \leftarrow t + 1$

For an agent controlled by the metacontrol policy π, the reward the agent is expected to accumulate starting from a state $s \in \mathcal{S}$ is defined as:

$$V^\pi(s) = E_\pi \left[\sum_{t'=t}^{T-1} r_{t'} | s_t = s \right] \quad (1)$$

where T is the time point when the agent enters a terminal state $s_T \in \mathcal{S}^\dagger$. The expected life-time reward performance measure is then $V^\pi(s_0)$.

3 Related Work

Stochastic Shortest Path. If there is no death state, then optimizing the expected time to reach the goal state (first performance measure above) is a stochastic shortest path problem (Bertsekas 1995). In the case when there is a non-zero probability of the agent reaching the death state under any metacontrol policy, a family of algorithms has been proposed (Kolobov, Mausam, and Weld 2012). The algorithms in Kolobov, Mausam, and Weld's approach extend an algorithm by Bonet and Geffner (2003) by adding to each iteration a step for eliminating *traps*: the states where taking the shortest path to the goal is likely to end in the death state.

The drawback of using these algorithms for metacontrol is that they require knowing the combined transition probabilities of the environment and the ground policy, which may not be readily available. For instance, in *Angband* the transition probability function $\mathcal{P}$ is encoded implicitly in some one hundred thousand lines of the game code. Likewise, the ground policy of one AI player, called `Borg` (White 1995), is some fifty thousand lines of C code.

Hierarchical Reinforcement Learning considers metacontrol at the level of *activities*: policies on subsets of the state space (levels in our case). The metacontrol learns a policy that maps the agent's states to these activities instead of primitive actions in the environment. At each point, the metacontrol selects an activity, and the agent acts according to that activity until it terminates and a new activity is selected. The MAXQ method (Dietterich 2000) describes each activity as a separate Markov Decision Process by defining

for each one a local reward function. The algorithm acquires the hierarchical policy by jointly learning a locally optimal policy for each activity with respect to these local reward functions. In the option framework of Sutton, Precup, and Singh (1999), each activity is defined as an option composed of a policy and a termination condition. The option policies are generally provided, or learned *a priori*, by treating each activity as a separate reinforcement learning problem. The Hierarchy of Abstract Machines (Parr and Russell 1998) defines each activity as a stochastic finite-state machine where state transitions cause actions to be taken in the environment.

One drawback of hierarchical reinforcement learning is that it requires an appropriate reward signal. Given several objectives, the scalar reward signal must encode an appropriate trade-off. Furthermore, the signal needs to be delivered fairly frequently for otherwise the agent is effectively random in the early stages of learning.

Metacontrol through Intrinsic Rewards. A curiosity-motivated AI agent learns a model of its environment. The original reward signal is then augmented with intrinsic *curiosity rewards* based on the current mismatch between predictions of the agent's model and reality. This type of metacontrol is based on the agent's assessment of its own knowledge. In the initial version, the curiosity rewards were directly proportional to the predictor error (Schmidhuber 1990). This leads the agent to levels where prediction error is high and the model needs improvement. A drawback of this approach is that the agent will also seek out the levels where the environment is less predictable (i.e., the prediction error is high even with the best predictor). Schmidhuber (1991) based the curiosity rewards on the change in the predictor error over time. This leads the agent to the levels where the predictor improves the most. Later work (Storck, Hochreiter, and Schmidhuber 1995) similarly introduced rewards based on information gain: the difference between the model's estimated transition probability distribution before and after a state transition is observed.

Bulitko and Brown (2012) introduced similar intrinsic rewards to reinforcement learning agents through a computational model of flow. In their framework, the degree of flow experienced by the agent was an intrinsic reward, much like the curiosity rewards. However, instead of augmenting the reward stream directly, their agents learned a value function for the expected extrinsic return and attempted to maximize a linear combination of such a value function and a flow reward (a reciprocal of the absolute difference between the agent's skill and the environmental difficulty).

Using intrinsic rewards in the ways described above is restricted to a reinforcement-learning setting. Additionally, augmenting the original reward signal with intrinsic rewards may make the new optimal policy suboptimal for the original problem (Ng, Harada, and Russell 1999; Wiewiora, Cottrell, and Elkan 2003).

Dynamic Difficulty Adjustment. Using a metacontrol policy to select a level in the environment is related to dynamic difficulty adjustment: the process of automatically changing aspects of a video game as it is played to avoid the player's becoming bored or frustrated with the game (i.e., to keep them in flow). The ground policy is a human player.

Many approaches use game features to predict the player's state. Hunicke and Chapman (2004) used probabilistic methods to predict inventory shortfalls (e.g., lack of specific weapons or ammunition) by observing trends in the damage taken by the playable character and inventory expenditure in a first-person shooter game. When shortfall is predicted, a handcrafted policy adjusts the game environment. Magerko, Stensrud, and Holt (2006) modeled players using a vector of competence levels for different skills. The approach individualizes training by finding the best match between characteristics of available training scenarios and the current state of the skill vector. Lankveld and Spronck (2008) measured skill in a role-playing game based on incongruity, the distance between the actual dynamics of the game and the mental model the player has built. They estimate this incongruity by considering the avatar's health relative to the progress made in the game, modifying game difficulty to enforce a given level of incongruity.

Similar techniques have been applied in commercial video games. For instance, the game *Left4Dead* adjusts its difficulty by estimating the emotional intensity of the players from their interaction with the game (Booth 2009). If the intensity is deemed too high, the enemy production is reduced, and major threats are not produced, lowering the game difficulty. Conversely, if the player's intensity falls too low, the enemy production is increased, increasing the challenge level of the game.

Other approaches learn from player feedback. Pedersen, Togelius, and Yannakakis (2009) trained a neural network to predict the player's ratings of fun, challenge, and frustration based on the player's behaviour and in-game content. Yannakakis, Lund, and Hallam (2007) used evolving neural networks to predict the player's self-reported interest based on game features. Zook and Riedl (2014) used tensor factorization to correlate time-varying measures of the player's performance in adventure role-playing games to the player's self-reported difficulty. This enabled forecasts of players' skill mastery by taking into account trends in the player's performance over time.

These approaches have the drawback of collecting and relying on reports from human players, usually in the form of user studies which can be cost-intensive. Another drawback of dynamic difficulty adjustment in general is that the techniques are usually tailored to improving the player's experience of the game (e.g., the feeling of enjoyment), which may not be the same as optimizing the player's performance (e.g., finishing a game level as quickly as possible).

4 Maximizing Flow as a Meta-Control

We will now introduce our approach to metacontrol based on maximization of flow. This is an extension of the work of Bulitko and Brown (2012), for multidimensional skills which allows us to consider as skills many different factors that impact the performance of a player.

4.1 Agent Skills

The goal of our metacontrol is to guide the agent to the right level of the state space given its skills. Skills are represented as a d-dimensional vector of real numbers; the

agent in state $s \in \mathcal{S}$ has skills $\bar{\sigma}(s) \in \mathbb{R}^d$. Notationally, $\bar{\sigma}(s) = (\sigma_1, \sigma_2, \ldots, \sigma_d)$, where each scalar σ_k represents a single skill. We abbreviate $\bar{\sigma}(s_t)$ to $\bar{\sigma}_t$ which represents the agent's skills at time t.

In a game such as *Angband*, given a fixed ground policy, the agent's skills can be described by the character's attributes. For instance, the agent's skills may be represented by the armor class (AC) of 10 and the hit points (HP) of 20 so that $\bar{\sigma} = (10, 20)$.

4.2 Level Complexity

Intuitively, the *difficulty* of a level in the environment is the minimum skill the agent needs to reliably survive the level and achieve the goal state (i.e., win). Algorithmically, the level's difficulty can be approximated by placing agents with widely different skills in the level and noting the ones that survive to the goal. Taking the per-component minimum of the survivors' skill vectors estimates the level's difficulty. To illustrate, suppose two agents whose skills at level 7 of *Angband* were $(10, 17)$ and $(8, 20)$ survived to the end of the game. Then the difficulty of level 7 would be estimated as $(\min\{10, 8\}, \min\{17, 20\}) = (8, 17)$.

If the environment is stochastic, then some agents may reach the goal purely by luck, lacking sufficient skills to do so reliably. We adjust for that by removing the bottom $\rho\%$ of the agents in each skill dimension. As an illustration, consider the plot in Figure 2 which shows a single skill value for all agents observed on level 5 of an environment. The agents that went on to reach the goal (i.e., won the game) are shown as black circles. To estimate the difficulty of level 5, we eliminate all agents who died before reaching the goal (red crosses on the left) as well as the bottom $\rho = 10\%$ of the winners, since they are deemed to have been underqualified and have won purely by luck (red crosses inside black circles on the right). The resulting difficulty estimate (16.65) is shown as the black horizontal bar in the figure.

The level difficulty is thus a bar below which any agent is deemed to be underqualified and having reached the goal by luck. We deem winners underqualified if the chances of having a skill value as low as theirs and yet reaching the goal are below a threshold ρ. Mathematically, the probability of having a skill value below a certain bar and yet reaching the goal are given by:

$$p_k^l(x) = \Pr\{\sigma_k^l < x \mid \text{agent reached the goal}\} \tag{2}$$

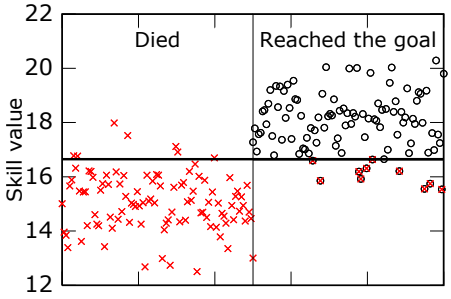

Figure 2: Level difficulty estimation.

where σ_k^l is the value of the k-th skill dimension the agent had at level l, and x is the position of the bar. We say that, if $p_k^l(x) < \rho$, then x is below the level difficulty and any agent with a lower skill value than x was underqualified, even if they reached the goal. Conversely, if $p_k^l(x) \geq \rho$, then x is above the level difficulty.

Thus, we define the level difficulty as the minimum skill value x where $p_k^l(x) \geq \rho$:

$$c_k(l) = \inf\{x \mid p_k^l(x) \geq \rho\}. \tag{3}$$

In the example in Figure 2, $k = 1$ and $l = 5$. The proportion of winners with a skill value below 16.65 at level 5 is 10% so we estimate that $p_k^l(16.65) = 0.1$.

The difficulty is estimated individually for each dimension k of the skill vector. The full difficulty of level l is then the vector $\bar{c}(l) = (c_1(l), \ldots, c_d(l))$, for a given threshold ρ.

4.3 Degree of Flow

Extending (Bulitko and Brown 2012; Bulitko 2014), we define the *degree of flow* experienced by an agent at time t on level l as:

$$F(\bar{\sigma}_t, \bar{c}(l)) = \frac{1}{||\bar{\sigma}(s_t) - \bar{c}(l)|| + \epsilon} \tag{4}$$

where $\epsilon > 0$ is a real-valued constant that bounds the flow and $||\cdot||$ is the Euclidean distance. The degree of flow, F, takes on a maximum of $1/\epsilon$ when the skills of the agent, $\bar{\sigma}_t$, and the difficulty of the level, $\bar{c}(l)$, are equal.

To illustrate, consider an agent in the game of *Angband* with two skills: armour class and hit points, $\bar{\sigma}_t = (10, 20)$. If the difficulty of level 1 is $\bar{c}(1) = (4, 20)$ while then the degree of flow experienced by the agent there is $1/(||(10, 20) - (4, 20)|| + \epsilon) = 1/(6 + \epsilon)$.

4.4 A Flow-maximizing Metacontrol Policy

Our flow-maximizing metacontrol policy operates as follows. When the metacontrol is called (line 3 in Algorithm 1), it considers all inter-level actions available to it and computes the degree of flow on the resulting level. It then selects the action that maximizes this degree of flow:

$$a = \operatorname*{argmax}_{a' \in A} F\left(\bar{\sigma}_t, \bar{c}(l_{a'})\right) \tag{5}$$

where $l_{a'}$ is the intended level of the inter-level action, a'.

5 Empirical Evaluation

We evaluated our flow-maximization metacontrol policy in the game of *Angband*, a dungeon crawler with well-defined discrete dungeon levels. Angband is a complex environment. Each dungeon level is 198×66 grid with about 1200 different terrain features, objects and monsters leading to an approximate upper bound of 10^{40438} number of states in the game. The number of actions depends on the state but on average the player has access to between 10 and 50 actions. However a metacontrol policy was responsible only for selecting whether to stay at the agent's current level or go to a higher-level dungeon. The ground policy, extracted from an existing AI *Angband* player, `Borg` (White 1995), was responsible for all other actions.

We defined the goal state as the entry state of dungeon level 30 since reaching the actual end of the game (level 100) was not achievable for the Borg. Only approximately 10% of Borg agents reach level 30.

5.1 Defining Agent Skills

The Borg agent uses 202 attributes of the environment and the agent to make decisions. 20 of those are used by the Borg's inter-level control code to decide on whether to advance to a higher-level dungeon. We used these 20 attributes for one of two skill sets in our experiment (Borg skillset).

The other skill set was determined using correlation feature selection (Hall 1999) from the 202 attributes. The procedure looked for the attributes that correlate with reaching the goal state but correlate weakly among themselves.

We ran 3000 Borg agents and recorded the values of the 202 attributes upon entry to each dungeon level. We also recorded whether the agent reached the goal or not. The data was input to Weka's CfsSubsetEval method (Hall et al. 2009), yielding a subset of 26 attributes, which comprised the second skill set (CFS skillset).

5.2 Computing Level Complexity

To estimate the difficulty of each level we first ran 833 Borg agents modified as follows. Each agent was guided by the Borg policy (both ground and metacontrol) until it reached a pre-specified level m. After that, the Borg's metacontrol was replaced with the always-go-to-a-higher-level-dungeon policy. The parameter m was selected uniformly randomly in $[0, 30]$. We then used the difficulty estimation procedure described in Section 4.2 with $\rho = 10\%$.

5.3 Experiments

We compared our flow-maximizing metacontrol policy (Section 4.4) using two different skill sets to two baseline policies: the full Borg agent and a random metacontrol (as detailed in Table 1).

We ran 750 agents for each of the four metacontrols. All four agent types used the same ground policy. Each trial consisted of an agent starting from the initial state of the game and proceeding until they died or reached the goal.

We used the three performance measures defined in Section 2: the number of steps to reach the goal, the failure rate, and the mean life-time score. We computed means for all three measures as well as dispersion values in the form of the 95% normal-based confidence intervals (CI).

Comparison of failure rates was done using a chi-squared test with Marascuilo *post-hoc* test (Marascuilo 1966). To compare the mean score and time steps among different metacontrols, we used a permutation F-test with along Holm-Bonferroni adjusted pairwise permutation *post-hoc* tests (Higgins 2004) using 5000 permutations chosen at random without repetition. For all statistical tests, the significance level was chosen *a priori* to be $\alpha = 0.05$.

Table 1: Metacontrol contestants.

Metacontrol	Description
FlowDesigned	Flow maximization with Borg skill set.
FlowAuto	Flow maximization with CFS skill set.
Borg	The Borg metacontrol.
Random	Uniformly random inter-level action selection.

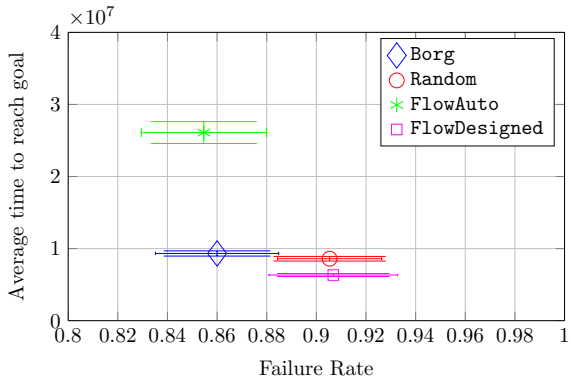

Figure 3: The average time (in steps) to reach goal and the failure rate (with 95% CI).

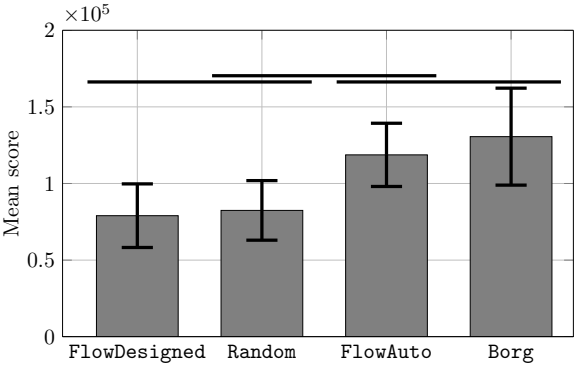

Figure 4: Mean score with the 95% CI. Horizontal lines at the top span the bars that were not significantly different from each other (pairwise permutation test).

5.4 Results

Metacontrol had a significant effect on both average time to reach the goal (permutation F-test, $p < 0.001$) and failure rate (chi-squared test, $\chi^2 = 15.15$, $df = 3$, $p = 0.0017$) as shown in Figure 3.

Pairwise permutation tests with Holm-Bonferroni adjusted alpha levels indicated that FlowDesigned agents reached the goal faster than texttttBorg agents and Random agents ($p < 0.001$). The FlowAuto metacontrol was significantly slower than the other three ($p < 0.001$).

The FlowDesigned metacontrol had a failure rate that was 6.28% higher than that of the FlowAuto metacontrol (Marascuilo critical range 5.58%) and 4.55% higher than that of the Borg metacontrol (Marascuilo critical range 4.36%). The failure rate of Random agents was also 5.87% higher than the FlowAuto agent failure rate (Marascuillo critical range 4.82%).

Metacontrol had a significant effect on the score (permutation F-test, $p = 0.012$) as shown in Figure 4. Pairwise permutation tests with Holm-Bonferroni adjusted alpha levels indicated that the average score of Borg agents was significantly higher than that of Random ($p = 0.0126$) and FlowDesigned agents ($p = 0.0054$). In addition, the average score of the FlowAuto agents was significantly higher than that of FlowDesigned agents ($p = 0.0084$).

5.5 Follow-up Experiments

Surviving FlowAuto agents took around 20 million steps longer to reach the goal than with other metacontrols. This may have been because they spent an average of 8.4 ± 2.5 million steps on level 7 far exceeding their overall per-level average of 0.5 ± 0.3 million steps. By contrast, Borg's average time on level 7 was only 0.66 ± 0.08 million steps.

This unusually long stay at level 7 appeared related to a decrease in level difficulty for the character strength skill from 18 (at level 7 and below) to 17 (at level 8). The decrease in the difficulty estimate goes against our assumption that higher skills are needed in higher levels. It is possible that we overestimate the difficulty of levels 7 and below due to the fact that all agents used to estimate the difficulty (Section 5.2) started their life with the skill value at 18. At level 8, their interactions with the game sometimes reduced their skill level to 17, which was then recorded as the difficulty.

The character strength is a skill that only the FlowAuto agents considered. It was not in the skill set of FlowDesigned which may explain the great difference in performance of the two flow-maximizing metacontrols. To investigate this possible difficulty overestimate, we artificially lowered the strength difficulty of levels 1 through 7 from 18 to 17. In a 300 trial experiment, the resulting metacontrol FlowAuto* became indistinguishable (based on 95% CI) from FlowDesigned on any measure (Table 2).

Additionally, preventing the Random metacontrol from descending to level 8 until at least 5 million steps after first entering level 7 (which we call slowed Random) decreased its failure rate by 8%, although the difference in score is not significant (based on 95% CI) (Table 3).

Together these two findings lead us to suspect that the better failure rate of FlowAuto relative to FlowDeigned is merely due to its unusually long stay on level 7.

Table 2: Mean steps to goal, failure rate, and score of the FlowAuto* and FlowDesigned metacontrols with 95% CI. The values for FlowAuto from the previous experiment are included for comparison.

	Mean steps [millions]	**Failure rate** [%]	**Score** [thousands]
FlowAuto*	6.14 ± 0.16	92.2 ± 3.0	61 ± 28
FlowDesigned	6.33 ± 0.20	90.7 ± 2.6	79 ± 21
FlowAuto	26.09 ± 0.65	85.5 ± 2.1	118 ± 21

Table 3: Mean steps to goal, failure rate, and score of the Random and slowed Random metacontrols with 95% CI.

	Mean time [millions]	**Failure rate** [%]	**Score** [thousands]
slowed Random	13.02 ± 0.35	82.0 ± 4.3	140 ± 40
Random	7.97 ± 0.47	90.0 ± 3.4	88 ± 30

5.6 Discussion

The results suggest that a flow-maximizing metacontrol is an improvement over a random metacontrol. Indeed, FlowDesigned metacontrol was faster and at least as reliable as Random. Likewise, FlowAuto metacontrol was more reliable and higher scoring than random but slower. However, the hand-engineered Borg metacontrol appeared to be the best. It was more reliable and higher scoring than FlowDesigned and faster and at least as reliable as FlowAuto.

The FlowAuto agents showed the most improvement over the Random metacontrol, but this may be due merely to a decrease in the mined difficulty of a single skill from level 7 to level 8. We speculate that this decrease does not represent the actual difficulty profile in *Angband* and is an artifact of our difficulty estimation algorithm.

6 Future Work Directions

Future research will theoretically and empirically explore the properties of the domains in which matching the agent skills and the environment difficulty preserves an optimal policy. In Reinforcement Learning environments, this is connected to work on reward shaping (Ng, Harada, and Russell 1999; Wiewiora, Cottrell, and Elkan 2003).

Another direction is to improve the difficulty estimation algorithm. Its current per-component minimum assumes that skills are independent from each other. We have started a preliminary investigation into clustering of the skill vectors recorded from different agents and then selecting the per-component minimum only within each cluster.

One potential application of the method is for incremental evolution (Gomez and Miikkulainen 1997) where complex behavior is learned incrementally, by starting with simple behavior and gradually making the task more challenging. Our metacontrol policy could be used to decide when to increase the complexity of the task.

Finally, our definition of the flow degree is simplistic in treating all skills and difficulty dimensions equally. Future work will investigate more advanced computational models of flow (Moneta 2012). It will also apply flow-maximizing metacontrol to human *Angband* players to see if their feeling of flow is indeed improved with the metacontrol.

7 Conclusions

This paper considered a level-based environment and presented a metacontrol algorithm based on matching level difficulty with level skill, thereby maximizing the degree of flow the agent is experiencing. We evaluated the algorithm in the game of *Angband* by proposing automatic ways of defining the agent's skills and the environment difficulty. Flow-maximizing agents reached the goal the fastest but were no more reliable or higher scoring than a random metacontrol and worse than the existing hand-coded metacontrol. With a different skill set, flow-maximizing agents had the same reliability and scores as the existing hand-coded agents but were slower to reach the goal.

Aknowledgements

We are grateful to our collegues in the Intelligent Reasoning Critiguing and Learning (IRCL) Group at the University of Alberta for fruitful discussions. We appreciate funding from the National Research and Engineering Council of Canada.

References

Anderson, M. L., and Oates, T. 2007. A review of recent research in metareasoning and metalearning. *AI Magazine* 28(1):12.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*. Number v. 1 in Athena scientific optimization and computation series. Athena Scientific.

Bonet, B., and Geffner, H. 2003. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI International Joint Conference on Artificial Intelligence*, 1233–1238.

Booth, M. 2009. The ai systems of left 4 dead. In *Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE09)*.

Bulitko, V., and Brown, M. 2012. Flow Maximization as a Guide to Optimizing Performance: A Computational Model. *Adv. in Cognitive Systems* 2(239-256).

Bulitko, V. 2014. Flow for Meta Control. *CoRR* abs/1407.4.

Carli, M.; Fave, A. D.; and Massimini, F. 1988. The quality of experience in the flow channels: Comparison of italian and us students.

Cox, M. T., and Raja, A. 2007. Metareasoning: A manifesto. Technical report.

Csikszentmihalyi, M. 1975. *Beyond Boredom and Anxiety*. The Jossey-Bass behavioral science series. Jossey-Bass Publishers.

Cutler, A.; Astrand, A.; Marsh, S.; Hill, G.; Teague, C.; and Swiger, C. 1990. Angband. http://rephial.org/.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Gomez, F., and Miikkulainen, R. 1997. Incremental evolution of complex general behavior. *Adaptive Behavior* 5(3-4):317–342.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations* 11(1):10–18.

Hall, M. a. 1999. Correlation-based Feature Selection for Machine Learning. *Methodology* 21i195-i20:1–5.

Higgins, J. 2004. *An Introduction to Modern Nonparametric Statistics*. Duxbury advanced series. Brooks/Cole.

Hunicke, R., and Chapman, V. 2004. AI for dynamic difficulty adjustment in games. *Challenges in Game Artificial Intelligence AAAI ...* 91–96.

Kolobov, A.; Mausam; and Weld, D. S. 2012. A Theory of Goal-Oriented MDPs with Dead Ends. *CoRR* abs/1210.4.

Lankveld, G. V., and Spronck, P. 2008. Difficulty Scaling through Incongruity. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, 228–229.

Magerko, B.; Stensrud, B. S.; and Holt, L. S. 2006. Bringing the Schoolhouse Inside the Box - A Tool for Engaging Individualized Training. In *25th Army Science Conference*.

Marascuilo, L. A. 1966. Large-sample multiple comparisons. *Psychological bulletin* 65(5):280.

Mayers, P. L. 1978. *Flow in adolescence and its relation to school experience.* Ph.D. Dissertation, ProQuest Information & Learning.

Moneta, G. 2012. On the measurement and conceptualization of flow. In Engeser, S., ed., *Advances in Flow Research*. Springer New York. 23–50.

Nakamura, J. 1988. Optimal experience and the uses of talent.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, 278–287.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, 1043–1049. MIT Press.

Pedersen, C.; Togelius, J.; and Yannakakis, G. N. 2009. Modeling player experience in Super Mario Bros. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, 132–139.

Schmidhuber, J. 1990. A Possibility for Implementing Curiosity and Boredom in Model-building Neural Controllers. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, 222–227. Cambridge, MA, USA: MIT Press.

Schmidhuber, J. 1991. Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, 1458–1463 vol.2.

Storck, J.; Hochreiter, S.; and Schmidhuber, J. 1995. Reinforcement Driven Information Acquisition In Non-Deterministic Environments.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1–2):181–211.

White, A. P. 1995. Angband borg. www.innovapain.com/borg/.

Wiewiora, E.; Cottrell, G.; and Elkan, C. 2003. Principled methods for advising reinforcement learning agents. In *ICML*, 792–799.

Yannakakis, G. N.; Lund, H. H.; and Hallam, J. 2007. Modeling children's entertainment in the playware playground. In *Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games, CIG'06*, 134–141.

Zook, A., and Riedl, M. O. 2014. Temporal Game Challenge Tailoring. *Computational Intelligence and AI in Games, IEEE Transactions on* PP(99):1.