Feedback-Guided Intention Scheduling for BDI Agents

Michael Dann
RMIT University
Melbourne, Australia
michael.dann@rmit.edu.au

John Thangarajah
RMIT University
Melbourne, Australia
john.thangarajah@rmit.edu.au

Minyi Li
RMIT University
Melbourne, Australia
minyi.li@rmit.edu.au

ABSTRACT

Intelligent agents, like those based on the popular BDI agent paradigm, typically pursue multiple goals in parallel. An intention scheduler is required to reason about the possible interactions between the agent's intentions to maximize some utility. An important consideration when scheduling intentions is the user's preferences over the goals and the ways in which the goals are achieved. These preferences are generally unknown in advance, time-consuming to elicit, hard to model, and difficult to incorporate into an intention scheduler. In this paper, we present a Monte Carlo Tree Search based intention scheduler (`pref`-MCTS) that is able to learn the user's preferences over intention schedules via low-burden comparative-type queries. It incorporates the learned preferences in guiding the search, leading to execution policies that are optimized towards the user's preferences and expectations. We evaluate our approach using an artificial oracle that shows that `pref`-MCTS improves over state-of-the-art baselines, even when provided with a limited number of preference queries and noisy labels. We also conducted a user study and showed that `pref`-MCTS is able to learn user preferences and generate schedules that are preferred by the users in real-time.

KEYWORDS

Preference-based reasoning; Intention scheduling; Learning preferences.

ACM Reference Format:
Michael Dann, John Thangarajah, and Minyi Li. 2023. Feedback-Guided Intention Scheduling for BDI Agents. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Autonomous agents typically serve to automate tasks for some human users. The BDI (Belief-Desire-Intention) [17] model of agency is a popular and mature agent paradigm that is the basis of several agent programming languages, e.g., JACK [31], JadeX [16] and Jason [2]. BDI agents are characterized by goals that the agent aims to achieve, and plans that are recipes for how to achieve those goals. When an agent commits to achieving a goal, that goal becomes an *intention*, and a plan is chosen from a set of possible plans to achieve that intention.

A key feature of BDI agents is their ability to pursue multiple intentions simultaneously. This provides concurrency and flexibility in large scale applications. However, the interleaving of steps in different intentions may result in conflicts or synergies, e.g., where the execution of a step in one intention makes the execution of a step in another intention impossible. These interactions are important considerations when scheduling and interleaving intentions.

One significant line of work within BDI agent scheduling centres around using *goal-plan trees (GPTs)* [23] to reason about interactions. Various methods have been attempted, including summary-information-based scheduling [22, 24], coverage-based scheduling [25, 30], and Monte Carlo Tree Search (MCTS)-based scheduling [9–11, 32, 33]. None of this work, however, considers *preferences* over the goals and the ways in which the goals are achieved. Visser et al. [28, 29] use a summary-information based approach to schedule intentions such that some pre-specified preferences are satisfied. For example, a travel-assistant agent might attempt to satisfy preferences related to the mode of transport (e.g., train, plane, car) when booking transport for a holiday. However, this approach was simplistic and limited to pre-specified preferences over some fixed properties, such as mode of transport, class of accommodation, etc.

In contrast, user preferences over goal achievements can be complex, e.g., the preference of a goal achievement might be conditional on the achievement of another; the relative urgency of the goals might depend on the current context; and so on. User preferences are also time-consuming to elicit, difficult to capture in a preference language, and more importantly, users may not realise or be able to express their preferences until the scheduling task is in progress. Therefore, relying purely on humans to handcraft preferences or preference relations is impractical, especially in complex problem domains. However, this is an important challenge as we develop autonomous systems that assist humans.

We address these limitations and propose a novel MCTS-based intention scheduler, `pref`-MCTS, that: (a) learns the user's preferences over intention schedules during execution, via low-burden comparison-type queries; and (b) seamlessly incorporates the learned preferences into strategies that guide the search, leading to execution policies that are optimized toward the user's preferences and expectations. The preference learner and the Monte Carlo Tree Search component interleave with each other throughout the scheduling process, promoting mutual enhancement in both learning and scheduling performance.

We evaluate `pref`-MCTS in two ways: (i) via an artificial "oracle" function that mimics a human user; and (ii) via a study with real human participants. The use of an oracle function enables fine-grained control over the experimental parameters and facilitates the reproduction of results. Moreover, it allows us to stress test `pref`-MCTS in a range of scenarios with different query budgets and noisy preference labels. The results of this experiment show that, after a limited number of queries, `pref`-MCTS aligns better with the user's preferences than state-of-the-art baselines.

The human user study seeks to evaluate the practical effectiveness of `pref`-MCTS in a setting where the participants are asked

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

to cast their preferences in real-time. The results from this study show that pref-MCTS: (i) is able to generate outcomes that are strongly preferred by human users, even after limited comparison queries; (ii) is not burdensome and can be used in real-time; and (iii) can account for preference relationships that would otherwise be non-trivial to elicit from human users.

2 PRELIMINARIES AND RELATED WORK

In BDI-based agent programming languages (e.g. JACK [31], JadeX [16]), the behaviour of an agent is specified in terms of beliefs, goals, and plans. An agent has *goals* that represent states of the environment it aims to achieve. The agent's information about the environment (and itself) are modelled as *beliefs*; and *plans* are recipes for the agent to modify the environment in order to achieve its goals.

Plans are composed of steps, which are either primitive actions that directly change the agent's environment, or subgoals that are in turn achieved by other plans. This relationship between a top-level goal, its plans and subgoals defines a tree structure for each top-level goal, termed a goal-plan tree (GPT) [23, 24]. When the agent commits to a goal, an *intention* is formed that essentially instantiates a GPT for that goal.

2.1 Goal-Plan Trees

Intuitively, each GPT, t_i, captures the various ways in which the agent may achieve a particular goal. At the root is a goal node representing the top-level goal, and its children are plan nodes representing the potential plans to achieve that goal. The children of plan nodes may be action nodes or further goal nodes (representing subgoals). The next step pointer, x_i, stores the current state of progress. It initially points to the top-level goal, then updates according to the execution path chosen. If the next step is an action, it is performed in an atomic manner. If the next step is a subgoal, a (sub)plan is selected to achieve the subgoal, and the steps in the (sub)plan are then executed, in a similar manner. In essence, this corresponds to progressively selecting an execution path through the goal's GPT structure. For an extended definition of goal-plan trees, see [32].

2.2 Intention Selection

An agent will often pursue multiple intentions concurrently and the paths chosen to achieve these intentions can interact with each other, possibly hindering the achievement of some intentions, e.g., if two intentions compete for the same resource [26]. The *intention scheduling problem* is the problem of choosing a path through each GPT and finding a suitable interleaving of the steps in each of the paths, so as to maximise some utility. For instance, a common utility is to maximise the total number of goals achieved [9, 11, 22] or to balance the elapsed time among the goals achieved [32]. Visser et al. [28, 29] considered maximising a fixed set of pre-specified preferences, as mentioned in the introduction of this paper. In contrast, the approach we present in this paper does not know or assume preferences of the user but instead elicits preferences over potential trajectories during the scheduling process, as we detail ahead.

2.3 MCTS-based Intention Scheduling

Similar to Yao and Logan [32], our approach to intention scheduling is built around Single-Player Monte-Carlo Tree Search (SP-MCTS) [18]. The SP-MCTS-based scheduler takes four parameters as input: E, the current state of the agent's environment; $I = \{(t_1, s_1), ..., (t_n, s_n)\}$, a set of GPTs and their current step pointers; α, the number of node expansions (i.e. iterations) to be performed; and β, the number of simulations to be performed per node expansion.

Each node of the MCTS tree represents a state, with the root node (denoted by η_0) representing the current state. The edges in the search tree represent either the selection of a plan or the execution of an action, and connect the parent node representing the original state to the resultant child node. Starting from η_0, the algorithm iteratively builds a search tree based on stochastic simulations. Each iteration involves the following process:

Selection: The selection phase selects a leaf node η_e for expansion. The selection is based on the reward values backpropagated from simulations to each node η in the search tree, and adopts the UCB1 formula [5] to balance the exploration of less visited nodes with the exploitation of high reward nodes:

$$\mathtt{UCB1}(\eta) = \mathtt{s}(\eta) + c\sqrt{\frac{2 \ln N}{m_\eta}} \quad (1)$$

where m_η is the number of visits to node η so far, $\mathtt{s}$ is an aggregate score function that measures the average quality of all simulations passing through η, N is the total number of simulations performed so far, and c controls the degree of exploration. The selection starts from the root node η_0 and progressively selects the child node with the highest UCB1 value until a leaf node η_e is reached.

Expansion: Once a leaf node η_e is selected, a list of nodes representing the possible next steps from η_e are added as children of η_e.

Simulation: In this phase, a child node η_c of η_e is randomly chosen for simulation. The simulation starts from η_c and selects executable steps at random until no further action or plans can be executed, i.e., a terminal state is reached and a trajectory is generated. Each generated trajectory is associated with a performance vector $\mathbf{v} = (v_1, \cdots, v_p)$, which is used to calculate the trajectory's return. In much previous work on MCTS-based intention scheduling [9, 11, 32], $\mathbf{v}$ is a binary vector that captures the achievement status of all top-level goals in the environment, and the return is just the sum of this vector, i.e. the number of top-level goals achieved.

Back-propagation: After β simulations, the return of each generated trajectory is backpropagated to all nodes in the search tree that were traversed, so that the visit count and the aggregate score $\mathtt{s}$ can be updated accordingly.

After α iterations, the child node of η_0 with the best quality (according to $\mathtt{s}$) will be selected, and the corresponding $(t^*, s^*) \in I$ will be executed by the scheduler.

3 INTENTION SCHEDULING WITH FEEDBACK

Often, a user has different criteria to evaluate the quality of a trajectory, and these criteria might be complex. For example, the user might place more importance on some goals compared to others. Similarly, they might prefer certain goals to be achieved more quickly than others. These preferences might also be conditional, e.g., the achievement of a goal might only be desirable if some other goal was also achieved. However, most existing scheduling approaches are merely designed to maximise the number of goals achieved. While Yao and Logan [32] introduce a more sophisticated approach that aims to achieve a "fair" interleaving of plans (one that progresses all intentions at roughly the same speed), this does not account for situations where the user may *want* one goal to be prioritised over another.

Instead of assuming that an evaluation function is ready and given, we enhance the existing work by bringing a human into the loop so as to elicit their preferences. A key advantage of the proposed approach is that it makes no assumptions about the types of preferences that a user might have; so long as all relevant criteria are captured by the trajectories' performance vectors, the approach is capable of learning arbitrary preference functions.

3.1 Eliciting a Preference Function

To facilitate user-friendly and low burden preference elicitation, we apply Pairwise Comparison, where the user indicates their preference for one option over another. This is considered one of the lowest burden elicitation methods [14] and has been applied in various domains including but not limited to requirement engineering [1, 13], multi-objective optimisation [20], and dialogue system evaluation [34]. To elicit scheduling preferences, the user is presented with two trajectories, τ_k and τ_l (and their associated performance vectors $\mathbf{v}_k$ and $\mathbf{v}_l$), and prompted to indicate which one they prefer. Their responses are recorded in a database D of triples (τ_k, τ_l, μ), where μ is a distribution over $\{k, l\}$ that indicates the user's preference. If the user prefers one trajectory over the other, then μ puts all of its mass on that choice. If they regard the two trajectories as equally preferable, then μ is uniform. If they indicate that the two trajectories are incomparable, then the pair is ignored.

To give a concrete example, suppose that a user is asked to provide preferences for a robot that performs household chores. The performance vectors in this case might capture the times at which various tasks were completed, e.g., in trajectory τ_k, the dishes were washed by 9:30am and the vacuuming was done by 10:30am, while in trajectory τ_l, the vacuuming was done by 9:45am and the dishes were washed by 10:15am. If the user does not mind which order the tasks are completed in and only wants the combined chores to be completed as quickly as possible, they will prefer τ_l over τ_k, and thus $\mu(k) = 0$, $\mu(l) = 1$. If, on the other hand, they only want the dishes to be washed by 9.30am (ready for breakfast use), they will instead prefer τ_k over τ_l, i.e., $\mu(k) = 1$, $\mu(l) = 0$.

Our training approach aims to acquire a preference function that can be used to evaluate trajectories for an MCTS-based scheduler. We follow the Bradley-Terry model [3] for estimating preference functions based on pairwise preference responses. This model assumes that the probability of the user preferring one trajectory over

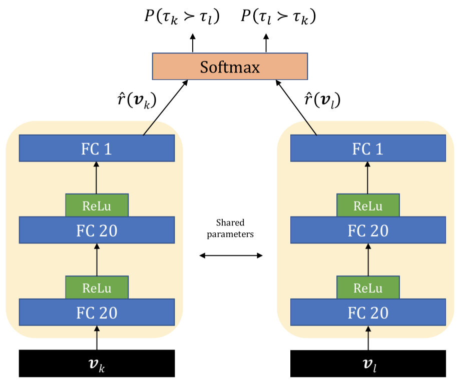


Figure 1: The architecture of the preference function.

another depends exponentially on the value of some latent preference function, $\hat{r}$, such that the probability of τ_k being preferred over τ_l is given by a softmax function:

$$P(\tau_k \succ \tau_l) = \frac{e^{\hat{r}(\mathbf{v}_k)}}{e^{\hat{r}(\mathbf{v}_k)} + e^{\hat{r}(\mathbf{v}_l)}} \tag{2}$$

where $\mathbf{v}_k$ and $\mathbf{v}_l$ are the performance vectors of τ_k and τ_l.

For those familiar with the chess rating system, or online games that have a ladder system, the value $\hat{r}(\mathbf{v}_i)$ returned by the preference function can be interpreted as an Elo rating (whose calculation is also based on the Bradley-Terry model [8]) or a match making rating (MMR) [27] for the performance vector $\mathbf{v}_i$.

To train the preference function from the database of user responses, we adopt the neural network architecture shown in Figure 1. The two parallel channels, highlighted in yellow, each take a performance vector $\mathbf{v}$ as input and return a preference score estimate $\hat{r}(\mathbf{v})$. Per the Bradley-Terry model, the preference score estimates are then fed through a softmax layer to determine the probabilities of the trajectories being preferred.

To ensure that the network's predictions are invariant to the order in which the trajectories are presented, we adopt a Siamese architecture [4, 15, 21] that shares parameters across the parallel channels. This means that, for example, if the network outputs probabilities of (90%, 10%) for input ($\mathbf{v}_k$, $\mathbf{v}_l$), it is guaranteed to output (10%, 90%) for the reversed input ($\mathbf{v}_l$, $\mathbf{v}_k$).

The network is trained by minimising the cross-entropy loss between the predicted preferences and the user's actual preference responses to the pairwise comparison queries:

$$loss(\hat{r}) = -\sum_{(\tau_k,\tau_l,\mu)\in\mathcal{D}} \mu(k) logP(\tau_k \succ \tau_l) + \mu(l) logP(\tau_l \succ \tau_k) \tag{3}$$

Since the number of pairwise comparisons is assumed to be tractable (we do not wish to burden the user with a large number of queries), we train from the full data set rather than performing a train-test split. We train for a fixed number of iterations and select the model with the least training error.

3.2 Incorporating Preferences into MCTS

We now turn to the task of incorporating the learned preference function into an MCTS-based scheduler, so as to schedule according to the user's preferences. We use Dann et al.'s [11] scheduler as a starting point, since its performance is state-of-the-art and its source code is publicly released.

The key modification we need to make is to change the aggregate score function, s, used in the selection phase of MCTS. For each node η under consideration, instead of calculating the average number of goals achieved (as in previous work that solely aims to maximise this quantity [9, 11, 22]), our preference-based scheduler calculates the average value of the preference function for all trajectories passing through η. That is, the aggregate score function s used in the UCB1 formula (Preliminaries, Equation 1) becomes:

$$\mathsf{s}(\eta) = \frac{\sum_{i=1}^{m} \hat{r}(\mathbf{v}_i)}{m} \tag{4}$$

where $\{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ are the performance vectors for all trajectories passing through η_e.

While this change in itself is straightforward, it introduces a subtle issue regarding the algorithm's exploration rate: In the UCB1 formula, the degree of exploration depends not only on the exploration constant, c, but rather on the relative scale of c and s. The larger the scale of the score function s, the larger the exploration constant must be. Since the scale of the learned preference function is dependent on the data set, this makes it difficult to pick a one-size-fits-all value for c. To address this, after training the preference function $\hat{r}$, we apply it to a fixed set of reference trajectories and normalise its output to have zero mean and unit variance. This allows us to achieve a similar degree of exploration across experimental domains while adopting a fixed value of c.

3.3 Iteratively Incorporating Feedback

One last detail that remains is the question of how the human user fits into the overall training procedure. Since the number of pairwise comparisons needed to learn an accurate preference function may be difficult to estimate upfront, we propose an iterative approach, as summarised in Algorithm 1.

To begin with, some default scheduler is used to generate an initial set of trajectories (lines 9–10). In our experiments, we use Dann et al.'s [11] unmodified scheduler as the default. The user is then asked to provide feedback on a number of trajectory pairs, drawn uniformly from the set of generated trajectories (lines 12–15). [1]This feedback is used to train a preference function (line 17), which is normalised using the default trajectories as a reference set (line 18). Lastly, the entire process is repeated iteratively, using the most recently trained preference function to perform scheduling.

The main benefit of this iterative approach is that the user can choose to terminate the process once they are satisfied with the performance of the scheduler, which keeps the amount of feedback required to a minimum. We also note that too much iteration may prove burdensome for the user, i.e., they may prefer to provide feedback in larger batches, which can be achieved by setting N_c to a larger value in Algorithm 1. We investigate the trade off between feedback frequency and scheduling quality in our experiments.

Algorithm 1 Overall Training Procedure

```
1:  var: max number of pairwise comparisons to gather, $N_{tot}$
2:  var: number of new trajectories scheduled per epoch, $N_t$
3:  var: number of new pairwise comparisons per epoch, $N_c$
4:
5:  Initialise trajectory set, T ← ∅
6:  Initialise training database, $D \leftarrow \emptyset$
7:
8:  while $|D| < N_{tot}$ and user is not satisfied do
9:      Schedule $N_t$ trajectories and append them to T. (Use
10:     default scheduler for the first iteration.)
11:
12:     for $i = 1$ to $N_c$ do
13:         Sample two trajectories $(\tau_k, \tau_l)$ uniformly from T.
14:         Elicit preference $\mu$ from user.
15:         $D \leftarrow D \cup (\tau_k, \tau_l, \mu)$
16:
17:     Train a preference function $\hat{r}$ from $D$.
18:     Normalise $\hat{r}$ w.r.t. default trajectories.
```

4 EVALUATION - TEST ORACLE

In order to evaluate our approach we conducted two experiments: (i) an empirical evaluation with the use of a test oracle; and (ii) an evaluation with human participants. We describe the evaluation with the test oracle here and the user study in the next section.

The aim of the first experiment was to answer the following questions:

(1) Is (`pref`-MCTS) capable of learning an accurate preference function from a tractable number of pairwise comparisons?
(2) How does varying the feedback frequency affect the performance of `pref`-MCTS?; and
(3) How robust is `pref`-MCTS to inconsistent feedback?

For this initial experiment, rather than enlisting human participants to judge trajectory pairs, we mimic the end user via an *oracle function* $\hat{o}(\mathbf{v})$ that maps a performance vector to a score value. This approach to evaluating preference learners is not new [7], and offers several advantages compared to enlisting human users:

- It allows for easier reproduction of results, which is necessary for others to continue work on this topic.
- It provides a natural performance measure for the experiments (how well the schedulers maximise the oracle score) and gives us a natural best-case baseline to compare against, as described shortly.
- With an oracle, Algorithm 1 can be run many more times than is feasible with human participants, allowing us to calculate granular results with tight error bounds. (In our later evaluation with human participants, we report only coarse, high-level findings.)

A further advantage of employing an oracle is that we can study the effect of inconsistent user feedback in a scientific way. To account for the fact that a user's judgements may not always be consistent with some internal score function, we incorporate noise into the labelling process. For each trajectory pair queried, we add Gaussian noise to the oracle scores, then determine the preference by comparing the noisy scores:

[1]Uniform sampling of trajectory pairs may appear simplistic, but in preliminary experiments it performed surprisingly well. This choice is discussed further in the appendix.

$$s_k = \hat{o}(\mathbf{v_k}) + n_k \sim \mathcal{N}(0, \sigma^2) \tag{5}$$

$$s_l = \hat{o}(\mathbf{v_l}) + n_l \sim \mathcal{N}(0, \sigma^2) \tag{6}$$

$$\mu(k) = \begin{cases} 0, \text{if } s_k > s_l \\ 1, \text{if } s_k < s_l \\ 0.5, \text{otherwise} \end{cases} \qquad \mu(l) = 1 - \mu(k) \tag{7}$$

4.1 Oracles Considered

Although much previous work on intention scheduling has focused on goal achievement, a user's preferences may depend on additional criteria. Hence we consider three different types of oracle function in our experiments:

- *Weighted goals*: The oracle scores depend solely on which goals were achieved, with different weightings given to the goals.
- *Contextual weighted goals*: Same as above, except that the goal weightings are permuted based on an environment variable, which in our experiments can take three possible values. For example, the environment variable might represent the weather (sunny / cloudy / rainy), where watering the garden has less value on a rainy day.
- *Time taken*: The oracle scores are a non-linear function of the time taken to achieve each goal.

The exact mathematical formulae for the oracle functions are provided in the appendix.

4.2 Baselines

We benchmark our scheduler against two alternatives:

- Dann et al.'s [11] scheduler, which strives to maximise goal achievement and was shown to outperform several competing schedulers at this task. Henceforth, we refer to this scheduler as the MCTS-`goals` baseline.
- A modified, "cheat" version of Dann et al.'s [11] scheduler that has direct knowledge of the oracle function and seeks to maximise it. We refer to this as the MCTS-`oracle` baseline.

Since the first of these schedulers also serves as the default scheduler for the first iteration of `pref`-MCTS, what one should expect to see is a learning curve where `pref`-MCTS initially matches the MCTS-`goals` baseline, then gradually improves towards the MCTS-`oracle` baseline.

4.3 Environment Details

To provide a thorough, varied evaluation of our approach, we employ a synthetic goal-plan tree generator. A fresh set of GPTs is generated at the start of each scheduling episode, so that the overall training task mimics a situation where the goals are semantically consistent across episodes, but the steps required to achieve them vary. We use the same GPT generator as employed in several recent papers [9, 11, 32]. Each scheduling task contains 10 GPTs, with 30 environment variables shared across the trees. Each GPT has a depth of 3, with 2 plans for every goal. Each plan contains 3 actions and 1 subgoal.

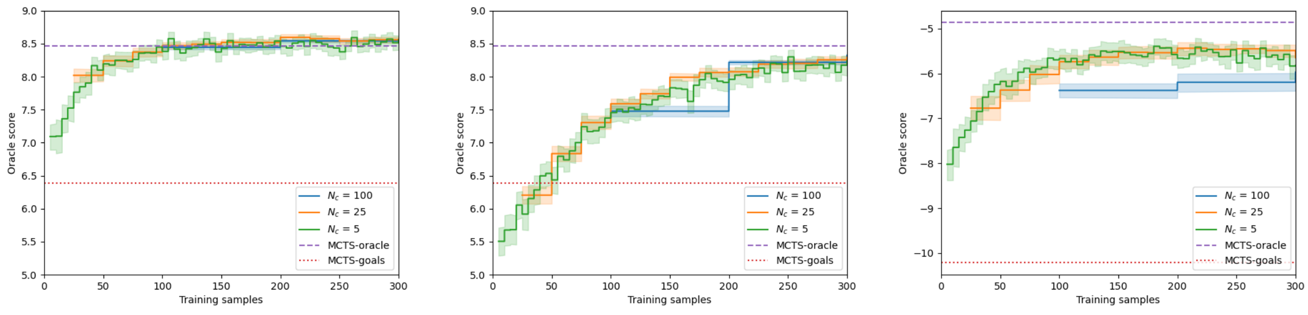
For the *weighted goals* and *contextual weighted goals* oracles, the tasks need to be difficult enough that the schedulers do not consistently achieve all goals, since this would render comparison impossible. Therefore, for experiments involving these oracles, we impose a time limit on the trajectories that makes it challenging to achieve all goals.

Full training hyperparameters (choice of optimiser, learning rate, etc.) are provided in the appendix. Source code is available at: https://github.com/mchldann/FeedbackGuidedIntentionScheduling.

4.4 Results

The learning progress of `pref`-MCTS under the three different oracle types is shown in Figure 2. In these experiments there is no noise added to the oracle scores; we investigate the impact of noise later. Dashed lines indicate the performance of the baselines (which do not learn, hence their performance is fixed), while each of the stepped curves shows the result for a different value of N_c (the number of new pairwise comparisons elicited per iteration). The schedulers are evaluated in terms of how well they maximise the oracle score, and each curve is averaged over 50 individual trials. Shaded regions indicate bootstrapped 95% confidence intervals, calculated per Seaborn's `lineplot()`.

Under the *weighted goals* oracle, which has the least functional complexity, `pref`-MCTS makes fast progress (Figure 2a). Its scheduling performance improves drastically after only tens of samples, and matches that of the MCTS-`oracle` baseline after around 80 samples.



(a) *Weighted goals* **(b) *Contextual weighted goals*** **(c) *Time taken***

Figure 2: Learning curves for `pref`-MCTS under different types of oracle function.

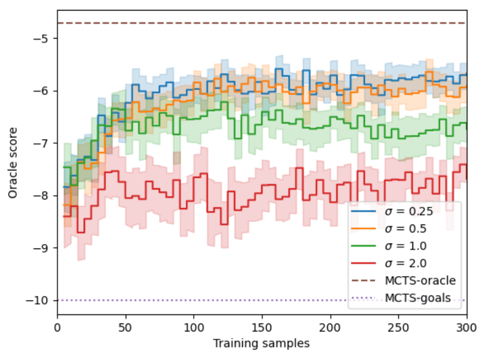


Figure 3: The impact of varying the noise level (σ) under the *time taken* oracle.

Modifying the oracle so that the goal weightings are contextual has a noticeable impact on sample efficiency (Figure 2b). Here, `pref`-MCTS takes around 250 samples to reach its peak performance, or roughly three times longer than under the previous oracle. This is a logical result, given that the context variable takes three possible values. It also suggests that our approach could run into sample efficiency issues if there were a much greater number of contexts. However, in most imaginable real-world scenarios, it is unlikely that *all* goal weightings would vary significantly under different contexts, as is the case here. Realistically, there are likely to be some goals that are important regardless of context, in which case the learned preference function ought to generalise at least partially. Indeed, this assumption is consistent with the results of our later user study conducted with human participants.

Also note that with the increase in functional complexity, `pref`-MCTS no longer quite reaches the performance of MCTS-`oracle`. However, it still improves significantly and closes most of the gap between MCTS-`goals` and MCTS-`oracle`.

The most striking result under the *time taken* oracle (Figure 2c) is that the feedback frequency has much more of an impact than in the previous experiments, with N_c = 100 significantly underperforming N_c = 5 and N_c = 25. This likely relates to the quality of the trajectories generated by the default scheduler (MCTS-`goals`) at the start of training: Under the previous oracles, there is a reasonable chance of MCTS-`goals` generating high-quality trajectories through chance, since the default scheduler used to generate the initial trajectories may luckily complete goals with large weightings. However, to generate a high-quality trajectory under the *time taken* oracle, the scheduler must achieve certain goals *quickly*. This is unlikely to occur through chance, because the default scheduler is not designed to optimise goal timings. Accordingly, the preference function is initially trained from a data set containing few high-quality trajectories, which in turn makes it a poor judge of such trajectories. Increasing the feedback frequency mitigates this problem, because `pref`-MCTS improves quickly and the data set does not become saturated with poor-quality trajectories.

Figure 3 shows the impact of adding noise to the training labels under the *time taken* oracle. Table 1 provides additional context, showing how the standard deviation of the noise affects the misclassification rate (the proportion of the training labels, μ, that are inverted by the noise). Note that the maximum possible misclassification rate due to noise is 50% (not 100%) because half the samples will still be classified correctly under random labelling.

Noise std. dev.	Misclassification rate
0.25	6.4%
0.50	10.9%
1.00	15.6%
2.00	25.1%

Table 1: Misclassification rates for the *time taken* oracle.

The results are essentially consistent with what one would expect; as the noise grows, the performance of `pref`-MCTS deteriorates. However, they also show that `pref`-MCTS can tolerate a reasonable degree of inconsistent labelling; until the misclassification rate reaches around 10–15%, its performance is barely affected. Moreover, while its performance degrades under heavy noise (σ = 2.0), it still outperforms the MCTS-`goals` baseline.

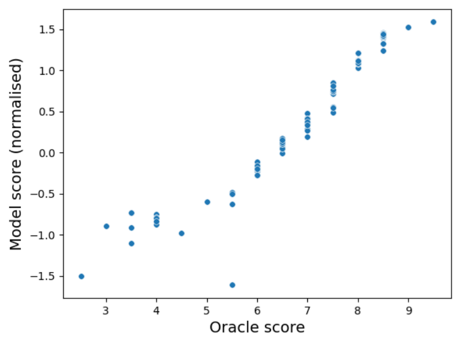
4.5 What Does the Preference Predictor Learn?

While pairwise comparison is a low burden method for preference elicitation, one of its limitations is that it cannot capture preference strength. For example, suppose that the user does in fact base their judgements on some internalised score value, similar to the oracle. If they consider trajectories τ_a, τ_b and τ_c to have scores of 0.1, 0.2 and 10.0 respectively, then they will express preferences of $\tau_c \succ \tau_b$ and $\tau_b \succ \tau_a$. However, this does not capture the fact that the former of these preferences is much stronger than the latter. Accordingly, the best we can hope for is that the learned preference function will be monotonic increasing with respect to the user's score. Of course, if it *were* practical to elicit numerical score values from the user then our approach could be simplified; one would just train directly from the numeric labels, rather than using the Bradley-Terry model. An implicit assumption throughout this work is that this is not feasible.

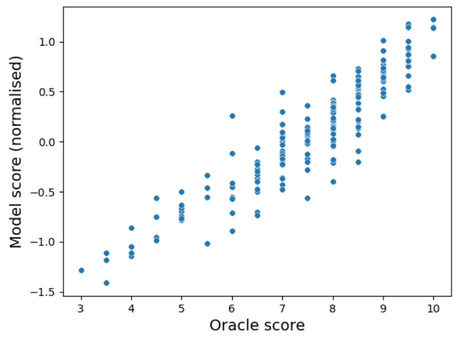
In light of the above discussion, we decided to investigate the shape of the function that `pref`-MCTS learns under different conditions by plotting the learned preference value versus the oracle score (Figure 4).

The function is taken at the end of training (after 300 samples) and each point represents one of the trajectories generated during training. The discrete vertical bands arise from the fact that the goal achievement oracles yield only a fixed number of distinct outputs.

Comparing Figure 4b to Figure 4a, the effect of adding a context variable is noticeable, with the learned function becoming more spread out and the greater overlap between bands signifying more prediction errors. While not shown here, a similar effect occurs when adding noise to the training labels. Interestingly, both plots have a roughly linear trend, despite what was noted above about the approach failing to capture the scale of preferences. However, this likely only reflects the fact that there are no major gaps in the score functions of oracles considered, in contrast to our extreme example, where there was a large jump in score from τ_b to τ_c.



(a) ***Weighted goals***



(b) ***Contextual weighted goals***

Figure 4: Learned score versus oracle score after 300 samples.

5 EVALUATION - USER STUDY

Whilst the above evaluation employing an artificial oracle allowed us to stress test `pref`-MCTS with complex functions and validate its effectiveness against the state-of-the-art of intention schedulers, we also wanted to test `pref`-MCTS with real human users to test its practical effectiveness in real-time.

5.1 Setup

For this study, we presented users with trajectories from a simplified version of the game *Super Mario World*[2], with an automated player controlling Mario. Mario has four goals: (i) complete level; (ii) find secret area; (iii) find Yoshi; and (iv) find mushroom. The performance vectors presented to the participants conveyed four pieces of information: (i) the goal completion statuses (ii); the level completion time (if applicable); (iii) the number of coins earned; and (iv) the number of enemies defeated. Figure 5 illustrates some of these characteristics. `pref`-MCTS was used to schedule the automated player's actions, with the human participants providing their feedback by comparing and selecting their preferred trajectories. We recruited 10 adults to participate in this study, comprising colleagues and students. All possessed a background in computer science but had limited knowledge of intention scheduling. Participants were not told any details about the underlying algorithm.

[2]https://en.wikipedia.org/wiki/Super_Mario_World

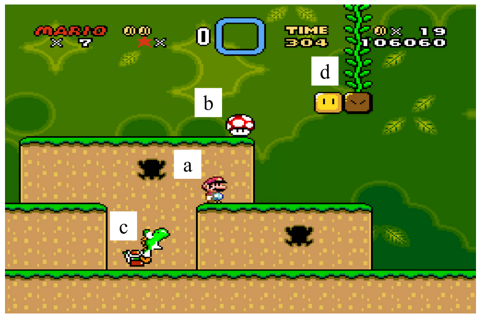


Figure 5: Super Mario World illustration - (a) Mario (b) Mushroom (c) Yoshi (d) Beanstalk leading to secret area.

5.2 Process

We first generate an *initial batch* of trajectories using the MCTS-`goals` scheduler, which is configured to maximise the total number of goals achieved, i.e., level completion, mushroom found, yoshi found, and secret area found.

The user is then prompted with 10 randomly sampled pairs of trajectory performance vectors, each specifying the goals achieved and the three quantitative variables: completion time, coins, and enemies defeated. For each pair, the user is asked whether they prefer one trajectory over the other, or whether they have equal preference over the pair. Figure 6 illustrates a sample preference query. We also ask and note a brief rationale for the choice.

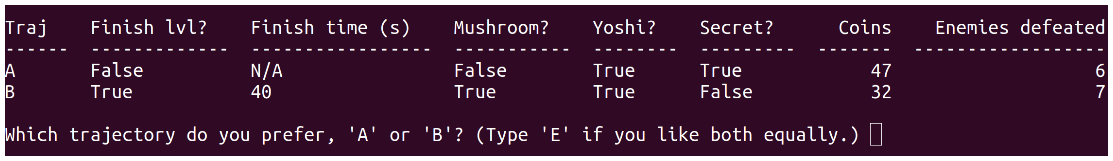
Based on the feedback received, `pref`-MCTS generates a *first batch* of trajectories incorporating user preferences. Another 10 pairs are then sampled and presented to the user for a second round of preference collection, following the same procedure as in the first round. Using this second round of feedback, `pref`-MCTS generates a *second batch* of trajectories incorporating user preferences.

Finally, users are prompted to compare the initial batch of sample trajectories with the trajectory samples generated after the first and the second iterations of user feedback collection (i.e., first batch and second batch), respectively. We do this by presenting the users with the average achievements of each trajectory batch, including: (i) the percentage of level completion; (ii) the percentage of trajectories where the secret area was found; (ii) the percentage of trajectories where Yoshi was found; (iv) the percentage of trajectories where the mushroom was found; (v) the average time use; (vi) the average number of coins earned; and (vii) the average number of enemies defeated. Figure 7 shows an example comparison between trajectory batches generated in different iterations.

Users were asked whether they prefer the initial batch of trajectories or the preference-based trajectories generated after each feedback iteration, and to what extent they prefer or do not prefer one over the other: slightly or strongly. We also asked for a brief rationale for why they would prefer one batch over the other.

5.3 Results

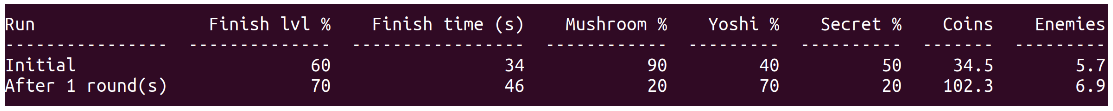
The results of the qualitative evaluation of the preference-based trajectories compared to the initial batch are presented in Figure 8. All of the users preferred the trajectories generated by `pref`-MCTS over the initial batch, with a majority (70%) strongly preferring the



```
Traj    Finish lvl?    Finish time (s)    Mushroom?    Yoshi?    Secret?      Coins    Enemies defeated
------  -------------  -----------------  -----------  --------  ---------  -------  ------------------
A       False          N/A                False        True      True            47                   6
B       True           40                 True         True      False           32                   7

Which trajectory do you prefer, 'A' or 'B'? (Type 'E' if you like both equally.)
```

Figure 6: Sample performance trajectory pair shown to the user when requesting feedback.



```
Run                Finish lvl %    Finish time (s)    Mushroom %    Yoshi %    Secret %    Coins    Enemies
---------------  --------------  -----------------  ------------  ---------  ----------  -------  ---------
Initial                      60                 34            90         40          50     34.5        5.7
After 1 round(s)             70                 46            20         70          20    102.3        6.9
```

Figure 7: Sample comparision query of batches of trajectories with average performance values.

preference-based batches. This validates the practical usefulness and effectiveness of our `pref`-MCTS scheduler.

The results show that `pref`-MCTS was able to learn preferences quickly, generating strongly preferred outcomes after just one round of limited (10) preference queries. On the other hand, the strong outcomes from the first batch meant that there was no noticeable improvement from the second batch, since the scheduler's performance was already strong.

We note that for one of the three participants who 'slightly preferred' the first batch over the initial batch, the initial batch of trajectories already achieved a rate of 80% level completion, due to the randomness of the trajectory generation. Level completion was a key preference of that user, hence it was hard to for them to see a strong improvement.

We also note that the users were able to cast their preference votes quickly, with the total time for evaluating 20 pairs of trajectories (over the two iterations) taking less than 5 minutes to complete. This indicates that the comparison queries are not burdensome and that the entire algorithm can be run in real-time.

We analyzed the user's responses and their rationale for their preference choices to gain further insight. It was interesting to observe that, despite the many possible combinations of preferences over the multiple criteria, human preferences, though diverse, were often simple in contrast to the preference functions of the artificial oracle from the previous section. This may help to explain why the learning curve in our user study appeared to be very sharp. Level completion was a common high priority (though not always the highest one), while preferences over the remaining factors were much more diverse. Some users prioritised 'speed running', i.e., finishing the level in as little time as possible, some preferred finding Yoshi, and many preferred multiple criteria together (e.g., maximising both the number of coins earned and enemies defeated). Eliciting diverse and multifaceted preference relationships is non-trivial and difficult to achieve via handcrafted rules, but our approach is able to account for such preferences effectively via low-burden comparison queries.

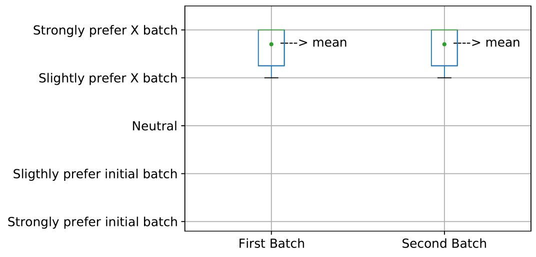


Figure 8: Results of user preference study, comparing the initial batch with `pref`-MCTS-generated batches.

6 CONCLUSION

In this paper we proposed `pref`-MCTS, an MCTS-based intention scheduler that learns user preferences through iterative, low-burden queries. The intentions are then scheduled to optimise towards those preferences.

We stress tested `pref`-MCTS using an artificial oracle, both to enable reproduction of results and to allow the experimental parameters to be fine-tuned. We tested different types of preference functions and introduced different levels of noise in the simulated human responses. The results showed that `pref`-MCTS improves over state-of-the-art baselines, even when provided with a limited number of preference queries, and that it is also reasonably tolerant to noisy preference responses.

We also conducted a user study with human participants and showed that `pref`-MCTS is able to learn user preferences from a limited number of comparison queries in real-time. The trajectory comparisons were not burdensome and the approach was able to account for diverse preference relationships based on a combination of factors. It would be impractical to elicit and precisely represent such preferences prior to scheduling, for example, as done in previous work by Visser et al. [28, 29].

Whilst `pref`-MCTS in this work has focussed on BDI agents, it could also be applied to other types of systems such as HTN planners that have similar features to BDI systems (see [12] for a comparison) and generally require preferences to be pre-specified in order for the planner to reason over them [19].

There are a number of potential directions for future work, including but not limited to: i) exploring different types of preference queries [6], e.g., point-wise (seeking an absolute preference score for a trajectory), list-wise (seeking preferences over multiple trajectories), etc., and ii) utilising historical preference data from similar domains to bootstrap the learning mechanism.

REFERENCES

[1] Yvonne Bijan, Junfang Yu, Jerrell Stracener, and Timothy Woods. 2013. Systems requirements engineering—State of the methodology. *Systems Engineering* 16, 3 (2013), 267–276. https://doi.org/10.1002/sys.21227 arXiv:https://incose.onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21227

[2] Rafael Bordini, Jomi Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Vol. 8. https://doi.org/10.1002/9780470061848

[3] Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika* 39, 3/4 (1952), 324–345. http://www.jstor.org/stable/2334029

[4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using a "Siamese" Time Delay Neural Network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems* (Denver, Colorado) *(NIPS'93)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 737–744.

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[6] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning* (Corvalis, Oregon, USA) *(ICML '07)*. Association for Computing Machinery, New York, NY, USA, 129–136. https://doi.org/10.1145/1273496.1273513

[7] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

[8] Rémi Coulom. 2007. Computing Elo ratings of move patterns in the game of Go. In *Computer games workshop*.

[9] Michael Dann, John Thangarajah, Yuan Yao, and Brian Logan. 2020. Intention-Aware Multiagent Scheduling. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, and G. Sukthankar (Eds.). 285–293.

[10] Michael Dann, Yuan Yao, Natasha Alechina, Brian Logan, and John Thangarajah. 2022. Multi-Agent Intention Progression with Reward Machines. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, Luc De Raedt (Ed.). ijcai.org, 215–222. https://doi.org/10.24963/ijcai.2022/31

[11] Michael Dann, Yuan Yao, Brian Logan, and John Thangarajah. 2021. Multi-Agent Intention Progression with Black-Box Agents. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Zhi-Hua Zhou (Ed.). International Joint Conferences on Artificial Intelligence Organization, 132–138. Main Track.

[12] Lavindra de Silva and Lin Padgham. 2004. A Comparison of BDI Based Real-Time Reasoning and HTN Based Planning. In *AI 2004: Advances in Artificial Intelligence, 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia, December 4-6, 2004, Proceedings*. 1167–1173.

[13] Georgia Dede, Persefoni Mitropoulou, Mara Nikolaidou, Thomas Kamalakis, and Christos Michalakelis. 2020. Safety requirements for symbiotic human–robot collaboration systems in smart factories: a pairwise comparison approach to explore requirements dependencies. *Requirements Engineering* 26 (2020), 115–141.

[14] Richard D. Goffin and James M. Olson. 2011. Is It All Relative? Comparative Judgments and the Possible Improvement of Self-Ratings and Ratings of Others. *Perspectives on Psychological Science* 6, 1 (2011), 48–60. http://www.jstor.org/stable/41613423

[15] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, Vol. 2. Lille.

[16] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. 2005. Jadex: A BDI Reasoning Engine. In *Multi-Agent Programming: Languages, Platforms and Applications*, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni (Eds.). Springer US, Boston, MA, 149–174. https://doi.org/10.1007/0-387-26350-0_6

[17] A. S. Rao and M. P. Georgeff. 1991. Modeling rational agents within a BDI-architecture. In *Principles of Knowledge Representation and Reasoning. Proceedings of the second International Conference*. Morgan Kaufmann, San Mateo, 473–484.

[18] Maarten P.D. Schadd, Mark H.M. Winands, Mandy J.W. Tak, and Jos W.H.M. Uiterwijk. 2012. Single-player Monte-Carlo tree search for SameGame. *Knowledge-Based Systems* 34 (2012), 3–11. https://doi.org/10.1016/j.knosys.2011.08.008 A Special Issue on Artificial Intelligence in Computer Games: AICG.

[19] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. 2009. HTN Planning with Preferences. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. 1790–1797.

[20] Kendall Taylor, Huong Ha, Minyi Li, Jeffrey Chan, and Xiaodong Li. 2021. Bayesian Preference Learning for Interactive Multi-objective Optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2021)*. Association for Computing Machinery, 466–475.

[21] Gerald Tesauro. 1989. Connectionist Learning of Expert Preferences by Comparison Training. In *Advances in Neural Information Processing Systems*, D. Touretzky (Ed.), Vol. 1. Morgan-Kaufmann.

[22] John Thangarajah and Lin Padgham. 2011. Computationally Effective Reasoning About Goal Interactions. *J. Autom. Reasoning* 47 (06 2011), 17–56. https://doi.org/10.1007/s10817-010-9175-0

[23] John Thangarajah, Lin Padgham, and Michael Winikoff. 2003. Detecting & Avoiding Interference Between Goals in Intelligent Agents. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (11 2003), 721–726.

[24] John Thangarajah, Lin Padgham, and Michael Winikoff. 2003. Detecting & Exploiting Positive Goal Interaction in Intelligent Agents. *Proceedings of the Interantional Conference on Autonomous Agents* 2, 401–408. https://doi.org/10.1145/860575.860640

[25] John Thangarajah, Sebastian Sardina, and Lin Padgham. 2012. Measuring Plan Coverage and Overlap for Agent Reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (Valencia, Spain) *(AAMAS '12)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1049–1056.

[26] John Thangarajah, Michael Winikoff, Lin Padgham, and Klaus Fischer. 2002. Avoiding Resource Conflicts in Intelligent Agents. In *ECAI '02: proceedings of the european conference on artificial intelligence*. Lyon, France, 18–22.

[27] Maxime Véron, Olivier Marin, and Sébastien Monnet. 2014. Matchmaking in multi-player on-line games: studying user traces to improve the user experience. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. 7–12.

[28] Simeon Visser, John Thangarajah, and James Harland. 2011. Reasoning about Preferences in Intelligent Agent Systems. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. IJCAI/AAAI, Barcelona,Spain, 426–431.

[29] Simeon Visser, John Thangarajah, James Harland, and Frank Dignum. 2016. Preference-based reasoning in BDI agent systems. *Autonomous Agents and Multi-Agent Systems* 30, 2 (2016), 291–330.

[30] Max Waters, Lin Padgham, and Sebastian Sardiña. 2014. Evaluating coverage based intention selection. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, Ana L. C. Bazzan, Michael N. Huhns, Alessio Lomuscio, and Paul Scerri (Eds.). IFAAMAS/ACM, 957–964. http://dl.acm.org/citation.cfm?id=2617398

[31] Michael Winikoff. 2005. *Jack™ Intelligent Agents: An Industrial Strength Platform*. 175–193. https://doi.org/10.1007/0-387-26350-0_7

[32] Yuan Yao and Brian Logan. 2016. Action-Level Intention Selection for BDI Agents. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems* (Singapore, Singapore) *(AAMAS '16)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1227–1236.

[33] Yuan Yao, Brian Logan, and John Thangarajah. 2016. Robust Execution of BDI Agent Programs by Exploiting Synergies between Intentions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 2558–2564.

[34] Wangchunshu Zhou and Ke Xu. 2020. Learning to compare for better training and evaluation of open domain natural language generation models. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI'20, Vol. 34)*. AAAI Press, 9717–9724.