An Adversarial Algorithm for Delegation

Juan Afanador and Murilo Baptista and Nir Oren

University of Aberdeen, AB24 3UE, Scotland
r01jca16,m.baptista,n.oren@abdn.ac.uk

Abstract. Task delegation lies at the heart of the service economy, and is a fundamental aspect of many agent marketplaces. Research in computational trust considers which agent a task should be delegated to for execution, given the agent's past behaviour. However, such work does not consider the effects of the agent delegating the task onwards, forming a chain of delegations before the task is finally executed (as occurs in many human outsourcing scenarios). In this paper we consider such delegation chains, and empirically demonstrate that existing trust based approaches do not handle these situations as well. We then introduce a new algorithm based on quitting games, to cater for recursive delegation.

1 Introduction

Agents seeking to achieve some goal may delegate tasks to others. Such delegations seek to increase the likelihood of the task being successfully executed, given the presumption that the agent receiving the task (the *delegatee*) is willing and capable to do so, on the part of the agent delegating the task (the *delegator*). While this is the commonly adopted view, the delegatee may actually not be the best suited agent for executing the task, but rather be able to further delegate (due to its knowledge or connections) to others who are. This type of *recursive delegation* has — to our knowledge — rarely been considered in the multi-agent systems community, though it captures a common situation where, e.g., projects are repeatedly contracted and subcontracted within organisations.

We believe that existing approaches to trust are ill-suited to making delegation decisions in domains where recursive delegation is possible. This arises due to several factors, namely that 1) agents within such a system are faced with a choice of whether to execute a task, or delegate it onwards; 2) delegators must learn about the competencies of their neighbours with respect to both delegation and execution; and 3) the topology of the network of possible interactions may change. The likelihood of a task being successfully executed thus depends on multiple conditions, resulting in potentially large changes in the likelihood of successful task execution, which are difficult to handle.

In this work, we propose an algorithm that explicitly considers recursive delegation by building on quitting games [15]. We then compare the performance of this algorithm to several existing techniques, empirically demonstrating its improved behaviour. Critically, we do not consider reputation, but only direct trust observations, meaning that evaluating our algorithm against many existing trust and reputation based approaches would be inappropriate. Instead, our evaluation concentrates on trust-based approaches for partner selection based on multi-armed bandits, namely an ϵ-greedy approach [17], UCB1 [2], Thompson Sampling [5], and the Gittins Index [7]. We describe these approaches in Section 2. In Section 3, we present our new quitting game based algorithm, providing an empirical comparison between the various approaches in Section 4. We discuss our results and situate them within existing work in Section 5, before concluding in Section 6.

2 Background

The problem of task delegation among partners with unknown competencies can be viewed as an exploitation/exploration problem, where partners should have tasks delegated to them (exploitation), while unknown agents should occasionally have tasks delegated to them so as to determine their competence (exploration). A common framework for modelling, precisely, this class of problems is offered by multi-armed bandit models, or multi-armed bandits (MABs) for short; an overview of which will be provided in what follows, accompanied by the algorithms used to solve them.

2.1 Multi-Armed Bandits

A multi-armed bandit problem depicts a scenario where a single agent must repeatedly select one among several courses of action, obtaining a reward from this action. The repeated occurrence of an action can affect the rewards it yields, an effect modelled by a random variable which — whenever the action is performed — can cause a change to occur in the reward state underpinning the action. In the MAB model, each potential action is referred to as an *arm*, while choosing the action is referred to as *pulling an arm*.

Definition 1 (Multi-Armed Bandits — Arms). *An arm A is a tuple $\langle X, r, h, f\rangle$ where X is an ordered list of possible states of the arm, and r is a probability distribution over possible rewards, parameterised by X.*

The history *of the arm, h, is a set of pairs (x_h, l_h) where $l_h \in \mathbb{Z}$ is the number of times the arm was pulled while in the state indexed by x_h. The* current state *of the arm is the state associated with the largest index of the arm's history with a non-zero l_h.*

Denoting the set of all possible histories as H, and the index of the current state of the arm as x, f is a probability distribution over the states $[x_h, x_{h+1}]$ parameterised over H.

Definition 2 (Multi-Armed Bandits — Pulling an arm). *Pulling an arm with current state x_i and history $h = [(x_1, l_1), \ldots, (x_i, l_i), (x_{i+1}, 0), \ldots (x_n, 0)]$ will update the arm's history to h' as follows:*

$$h' = \begin{cases} [(x_1, l_1), \ldots, (x_i, l_i + 1), (x_{i+1}, 0), \ldots (x_n, 0)] & \textit{if } f(h) = x \\ [(x_1, l_1), \ldots, (x_i, l_i), (x_{i+1}, 1), \ldots (x_n, 0)] & \textit{otherwise} \end{cases}$$

A multi-arm bandit is, then, a set $\mathcal{A}$ of arms. The number of times each arm was pulled starts at zero. Pulling an arm updates the arm as described above, and — given that the arm is in state x, yields a reward R with likelihood $r(x, R)$.

A policy is a function $\mathcal{S} : [a_1, \ldots, a_n] \times [r_1, \ldots, r_n] \rightarrow \mathcal{A}$. In other words, given a sequence of arm pulls and the rewards obtained, the policy specifies which arm should be pulled next. The main problem considered by the MAB literature involves identifying a policy which is in some sense optimal, e.g., which maximises rewards, or minimises regret. It has been long established that if the MAB's states and the probability distribution of its rewards are known, the Gittins Index can be used to identify the optimal arm to pull [7].

Formally, the Gittins Index for arm i in state x_i, with a discount factor for future rewards of β, is defined as follows:

$$G(x_i) = sup_{\sigma>0} \frac{E[\sum_{t=0}^{\sigma-1} \beta^t r(x_i)| \text{ initial state of arm}]}{E[\sum_{t=0}^{\sigma-1} \beta^t | \text{ initial state of arm}]}$$

The Gittins Index computes the expected reward of pulling arm x_i against the cost of not pulling it, and thus identifies the arm with the highest expected reward as the one that should be pulled. Calculating the Gittins Index is computationally prohibitive [7], in response to which various numerical approximations have been proposed in the literature [3, 8].

More importantly, in practice, the probability distribution of the rewards and the states of each arm may not be known. In this case, the Gittins Index may be used as a heuristic based on beliefs about rewards and arm states, which means that different ways of calculating these beliefs will result in different procedures with very distinct properties. We now describe several such heuristics addressing the MAB problem, namely UCB1 [2], ϵ-greedy [17], and Thompson Sampling [5]. We will compare the performance of our approach to these heuristics in Section 4.

2.2 MAB Heuristics

We begin this section by briefly describing several well-known MAB heuristics in the context of standard MABs. In Section 3 we detail how these heuristics must be modified to deal with recursive delegation.

UCB1. Rather than simply maximising rewards, *upper confidence bound* (UCB) algorithms, exemplified by UCB1 [2], attempt to minimise decision-theoretic regret — the difference between the expected reward obtained had the optimal arm been pulled, and the expected reward of some other arm pulling policy. UCB1 is simple to implement and works well in practice, while guaranteeing that the achieved regret will grow only logarithmically with the number of arm pulls that occur.

For an arm j, UCB1 tracks the average reward obtained from that arm (μ_j), and the number of times the arm has been pulled (n_j), as well as the total number of times that the MAB's arms have been pulled (n). It then picks arm j, so as to maximise an upper bound on the mean expected reward given by the following equation [2]:

$$\mu_j + \sqrt{\frac{2 \ln n}{n_j}},$$

This choice guarantees that the probability of deviating from the population mean decays exponentially in time, in accordance with the Chernoff-Hoeffding inequality [10]. Once the arm has been pulled, μ_j, n_j and n are updated to identify the next arm to pull.

Thompson Sampling. This is another simple approach to selecting an arm, and does so by sampling an expected reward based on the arm's history, before selecting the arm whose sample reward is maximal. To perform such sampling, a probability distribution over the arms is required [1]. In this work we consider binary rewards, and we therefore perform our sampling using a Beta distribution, whose parameters record the number of times the arm returned a reward, and the number of times it did not. Thompson sampling then samples each arm using this probability distribution, and selects the arm which — using this sampling — has the highest expected reward.

ϵ-Greedy. This heuristic selects the arm which will yield the highest expected reward with likelihood $1 - \epsilon$ [17]. In the remaining cases, it will pick an arm at random. It is important to note that this heuristic differs from Thompson Sampling in that no sampling over the arms takes place, meaning that the best arm (in the sense of the expected reward) is always picked, unless a random arm is chosen (with likelihood ϵ).

All of the heuristics described above seek to balance exploitation (that is, selecting the arm most likely to give a high reward) with exploration (that is, learning more about the likelihood that the arms will give a reward). If the distribution governing the reward an arm gives is stationary, then these heuristics work well, and give well-understood convergence guarantees. However, in the case of recursive delegation, agents at each level learn simultaneously, meaning that the stationary distribution assumption is — until the learning stage ends — violated. It is for this reason that these heuristics function poorly when applied to recursive delegation. We conclude this section by briefly describing how we adapted the heuristics to operate in the domain of recursive delegation.

2.3 Applying MAB Heuristics to Recursive Delegation

Agents able to delegate to others must make two choices when tasked with an action, namely whether to execute the action themselves, or delegate it onwards (and in the latter case, must also decide who to delegate to). Each agent has a list of *delegatees* to which they can delegate a task. By viewing the delegatee agents as neighbours of the *delegator*, we obtain a directed graph over which a path represents a sequence of delegations.

We unify the execution/delegation decision for an agent by associating a *dummy agent* with each agent in the system, allowing the actual agent to delegate to the dummy agent, and ensuring that the dummy agent has no delegatee agents that they can pass the task onto. A task reaching the dummy agent must therefore be executed (by the agent associated with the dummy agent). Figure 1 illustrates a sample delegation network consisting of 6 agents ($a, \dots, f$), together with dummy agents ($a', \dots, f'$). In this scenario, one possible sequence of delegations (also referred to as a *delegation chain*) is a, b, c, f, f'

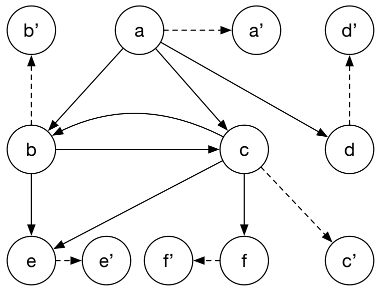


Fig. 1: A network of agents illustrating possible delegation links. Dotted lines indicate links to dummy agents which, when delegated to, execute the task.

To use the heuristics described above in a recursive context, agents make a *local* delegation decision, choosing who to pass the task to based only on their neighbours' potential to become delegatees. If a dummy agent receives the task, then it is executed, and feedback on success or failure is then provided to every agent in the delegation chain. Each agent then updates the statistics relevant to its delegation decision with respect to its neighbours, and the process repeats. Clearly, this approach prevents an agent from considering how others within the chain make decisions, and we claim that this affects the effectiveness of MAB heuristics in recursive delegation scenarios.

2.4 Quitting Games

We formulate an alternative approach to delegation which explicitly considers the actions available to agents through a game-theoretic mechanism based on quitting games [15]. Quitting games are multi-player stochastic games where players are faced with two choices, namely to *continue* (c) or to *quit* (q). The game ends and the players obtain rewards in two situations, whenever a *quit* action occurs, or the game reaches some terminal time. If the game does not end after the players have selected a move, i.e. simultaneous *continue* actions, then it enters another iteration where players act again, repeating this process until termination. Figure 2 illustrates a generic two-player quitting game between agents a and b.

The first entry in each terminal node appearing in Figure 2, corresponds to the reward accrued to a, the other denotes b's reward. Whenever (c_a, q_b) is played, a receives r_{c_a} and b obtains r_{q_b}, whereas (c_a, c_b) leads to yet unrealised rewards denoted by "$\circlearrowleft$". Agents a and b plan future moves by formulating *strategies* based on the anticipation of potential $\varepsilon - equilibria$.

Definition 3 (Quitting Game – Strategies). *At every iteration t within a time horizon T, each player i is provided with a set of actions $A_i = \{c_i, q_i\}$. A* strategy *is a probability measure $x^i_t : T \to [0, 1]$ denoting the likelihood of playing c_i at iteration t.*

Definition 4 (Quitting Game – ε-equilibrium). *A profile or vector of strategies $\boldsymbol{x}_t$, produces a stream of rewards r_{S_t}, contributed by those players S_t who have chosen not to quit the game, giving rise to an expected reward $v^i_t(\boldsymbol{x}_t) := \boldsymbol{E}_{\boldsymbol{x}}[r_{S_t} I_{t<\infty}]$. A solution*

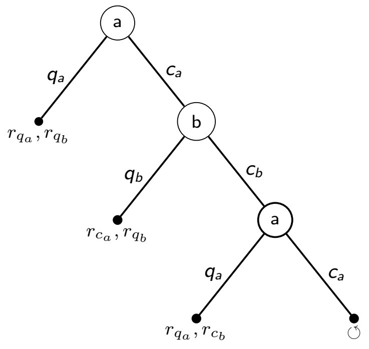


Fig. 2: Quitting Game in Extensive Form

concept *states the criteria for playing a particular profile. ε-equilibrium is the solution concept employed when solving a quitting game. A profile $\boldsymbol{x}_t$ is an ε-equilibrium if the expected reward it yields plus an overhead $\varepsilon_t > 0$, is at least that of any other strategy y_t^i for every player i:*

$$v_t^i(\boldsymbol{x}_t) \geq v_t^i(\boldsymbol{x}_t^{-i}, y_t^i) - \varepsilon_t.$$

Note that if $\varepsilon_t = 0$, the above expression produces a Nash equilibrium. ε-equilibria can be further qualified as *cyclic* if there exists a point in time $\tau \in T$ when $x_t^i = x_{t+\tau}^i$, or *stationary* if $x_t^i = x_0^i$ for each $t \in T$. For instance, given $r_{q_a} > 0$, $r_{c_a} < r_{q_b}$, $r_{q_a} < r_{c_a}$, and $r_{c_b} \geq r_{q_b}$, the stationary profile $(\mathbf{x}^a, c_b)$, $x_t^a \ll 1$ is an $\varepsilon - equilibrium$ of the game in Figure 2. More generally, every quitting game where players prefer unilateral termination to indefinite continuation, has a cyclic subgame perfect ε-equilibrium [15], while every two and three players quitting game has a stationary ε-equilibrium [16].

To use these ideas in the context of recursive delegation, a *delegator* playing a quitting game may never see the task executed, depending on the periodicity of the cyclic equilibrium, unless the delegation process unfolds either as a recursive negotiation with the same *delegatee*, or every delegation chain is constrained to no more than two *delegatees*. While most quitting games have more than 3 players in the context of recursive delegation – and we therefore have no guarantees regarding ε-equilibria – these games appear to capture an important aspect of recursive delegation. Therefore, the algorithm for recursive delegation which we propose in the next section builds on quitting games and, as discussed in Section 4, appears to outperform other MAB based approaches in this context.

3 Approach

As indicated in Section 2.1, the problem of task delegation may be seen as an exploitation/exploration problem in the spirit of MABs, where *delegators* waver between delegating the task to competent partners (exploitation) and delegating the task to unknown

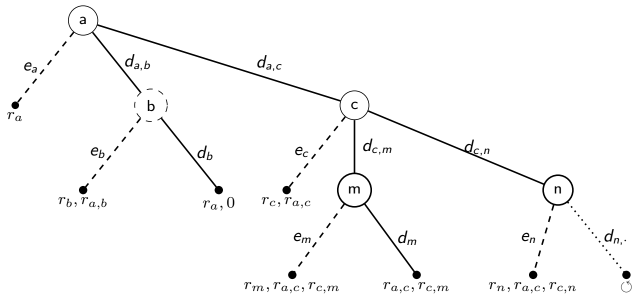


Fig. 3: Delegation Game in Extensive Form

partners (exploration). It is also apparent from Section 2.4 that recursive delegation has a natural predisposition to a game-theoretical treatment, due precisely to its explicit approach to recursion. In this section we present an algorithm for recursive delegation based on quitting games. Details on the corresponding adaptation of MAB heuristics, and the Gittins Index in particular, will be briefly addressed by the end of this section, procuring a comparable benchmark for Section 4.

3.1 Delegation as a Quitting Game

Quitting games are readily adaptable to recursive delegation. They typify the occurrence of self-embedded instances of strategic interaction, resembling the replication of delegation requests along a delegation chain. That is, if a *delegator* (a) and a potential *delegatee* (b) were to play a quitting game, to determine whether to delegate a task or not, the profile (c_a, c_b) would take them both to a new iteration of the same delegation request. Unlike a standard quitting game, however, a delegation process requires distinct strategic scenarios, where, e.g., b becomes a *delegator* facing a new *delegatee*. For this reason we have adjusted quitting games to this type of interactions, preferring instead the term *delegation games* when referring to them.

The players of a *delegation game* have a *delegate* (d) action and an *execute* (e) action, and their rewards depend on future *delegate* actions. Every pair of agents populating each instance of the game consists of one former *delegatee* acting as *delegator*, and one new agent serving as potential *delegatee*. *Delegation games* can only be prolonged by (d_i, d_j) profiles for every *delegator* i and *delegatee* j – provided there are available *delegatees* and sufficient time –, and are brought to an end whenever an *execute* action occurs. Future actions are formulated in terms of *strategies* and the pursue of $\varepsilon - equilibria$.

Definition 5 (Delegation Game). *A delegation game is a tuple* $\langle N, (A_i, u^i, r^i)_{i \in N} \rangle$. *All* N *agents, or players, pair up with each other in accordance with a predefined topology of interaction. A player generating a delegation request will be referred to as* delegator*, while a player at the receiving end of the delegation request will be termed* delegatee*. Potential* delegatees *within the reach of a* delegator *are said to be the latter's* neighbours.

Every iteration of the game comprises several instances of strategic interaction. There are as many instances in a single iteration, as available delegatees *can be found. At every iteration t within a time horizon T, each player i is provided with a set of actions $A_i = \{d_i, e_i\}$.*

Definition 6 (Delegation Game – Strategies and Expected Rewards). *A* strategy *is a probability measure $x^i_t : T \to [0, 1]$ indicating the likelihood of playing d_i at iteration t. Vectors of strategies $\boldsymbol{x}_t$ are termed profiles. $r^i_{D_t}$ is a random variable representing the rewards obtained from delegation by each player i, given the set of* delegatees *D_t at iteration t. $u^i_t : \boldsymbol{x}_{t-1} \times \mathbb{R} \to \Delta(A_i)$ is a measurable set-valued function that updates each player's strategies once an action e_j occurs or a terminal node is reached. Profiles induce a probability distribution which permits the computation of the expected rewards $v^i_t(\boldsymbol{x}_t) := \boldsymbol{E}_{\boldsymbol{x}}[r_{D_t} I_{t<\infty}]$.*

Figure 3 depicts one iteration of a (deterministic) *delegation game*. Agents a, b, c, m and n are arranged in a tree-like structure, where b and c are a's *neighbours*, m and n are c's *neighbours*, while b and m have no *neighbours*, and n is linked to another unspecified tree which allows delegation to continue. a has to decide between choosing a *delegatee* from $\{b, c\}$ or executing the task itself i.e. it has to decide whether to play $d_{a,b}$, $d_{a,c}$ or e_a.

Each one of the three branches radiating from a, in Figure 3, exemplifies an absorbing state of a *delegation game*. a can play e_a and perform the task itself. It can also delegate the task to b, in which case b might accept the task by playing e_b, or not by playing d_b, thus returning the task to a and forcing the occurrence of e_a. In each case, a and b receive $(r_a, 0)$, $(r_{a,b}, r_b)$ and $(r_a, 0)$, respectively. Alternatively, a could delegate to c. If n decides to play e_n, it receives r_n, while c and a obtain $r_{c,n}$ and $r_{a,c}$. The rewards of any agent in the delegation chain emanating from n's *neighbour*, will not be realised until some agent plays an *execute* action, the delegation process reaches a terminal node like b, or the time horizon is exhausted.

When rewards are subject to stochastic processes, the selection of an action has to be expressed in terms of strategic profiles ($\mathbf{x}_t$), as in Definition 6. The probability distribution these profiles induce is then used to calculate the expected rewards (v^i_t). By contrasting expected rewards in the manner of an ε-equilibrium, *delegators* and *delegatees* select a particular strategy, which once played provokes the respective information states to update (u^i_t). These ideas on how a *delegation game* operates, are presented in Algorithm 1.

In Algorithm 1, a set $B \subset N$ of neighbours is assigned to each of the N agents, and their respective rewards sampled from an uniform distribution (line 4). The resulting initial state allows the computation of individual mixed strategies i.e. the probabilities of delegating, whenever pairs of agents and neighbours engage in a delegation request (line 6). Note that the notation is preserved except for $r_{\cdot,1}$ and $r_{\cdot,0}$, denoting the rewards of executing the task given a *delegatee*'s willingness to further delegate or not. As long as there are neighbours who have not received such a request, despite holding a positive probability of delegating, the selection of the one with the highest expected pay-off will take place (lines 8 and 9), seeking a Nash equilibrium. If capable of executing the task, as given by a random "state of nature" (line10), this latter agent will have to weigh up the possibility of passing the task down the delegation chain or attempting its completion, thereby triggering a learning process (lines 11-15).

Algorithm 1 Delegation Game (DIG)

```
Input: $P := \{a_i, ad_i\}_{i \in N}$: Tuple of agents and their neighbours, $r$: Array of sampled rewards.
Output: $S$: Sequence of agents receiving a delegation request, $x$: Array of mixed strategies.
 1: function DIG($P_i$)
 2:     $S \leftarrow \{S_i\}_{i \in N}, x \leftarrow \{x_i\}_{i \in N}, r \leftarrow \{r_i\}_{i \in N}$
 3:     for j=1$\rightarrow N$ do
 4:         $ad_j \leftarrow \{a_k\}_{k \neq j \in B \subset N}, r_j \leftarrow \{\mathcal{U}(r_{j,0}, r_{j,T})\}_{j \in B \subset N}, S_j \leftarrow \emptyset, x_j \leftarrow 0$
 5:         for $a_k \in ad_j$ do
 6:             $x_{j,k} = \frac{r_{j,1} - r_{j,0}}{r_{j,k} - r_k}$
 7:             $x \leftarrow x \cup \{x_{j,k}\}$
 8:         while $(x \neq \emptyset) \wedge (\exists j[S_j == \emptyset])$ do
 9:             $m \leftarrow argmax_{j \in ad_j}(r)$
10:             if $(random() < x_{j,m})$ then
11:                 if $a_m \in S_j$ then
12:                     Update $x_{j,m}, r_{j,m}$
13:                 else $a_m \notin S_j$
14:                     $S_j \leftarrow S_j \cup \{a_m\}$
15:                 return LEARN($P_m; r_m, x_m$)
16:             else
17:                 $a_j$ executes the task
18:                 $S_j \leftarrow \emptyset$
19:     return $(S, x)$
```

```
1: function LEARN($P_i; r_i, x_i$)
2:     if $r_{i,0} \leq r_{i,1}$ then
3:         $a_i$ executes the task
4:         Update $x_{k,i}, r_{k,i}$
5:     else
6:         return DIG($P_i$)
```

3.2 Delegation as Nested MABs

We now specify a second heuristic which treats recursive delegation as a set of nested MABs, and where each agent makes a local decision regarding how to delegate based on an approximation to the Gittins Index. This heuristic is described in Algorithm 2.

Algorithm 2 is initialised in the same manner as Algorithm 1. It implements the Gittins Index through a beta reputation mechanism captured in lines 14-17, which feeds the numerical approximation to the index as specified in lines 6-8. The former is but a counter of successful delegation events, acting as a wrapper of the latter over recursive calls. In this way, monitoring behaviour is accounted for with a binary random variable keeping track of successful and failed choices.

The main procedure in Algorithm 2 is Brezzi and Lai's proposal of a MAB optimal policy. For a large number of trials, and a time-discounting rate $c \in [0.8, 1]$ – as calibrated by Brezzi and Lai [3] for efficient performance–, the following closed-form function is used to approximate the Gittins Index [3]:

$$G(T) \approx \mu + \sqrt{\frac{\mu(1-\mu)}{T+1}}\psi\left(\frac{1}{(T+1)^c}\right);$$

where μ is the mean of the compound distribution of the random variable indicating a successful delegation, and

$$\psi(t) = \begin{cases} \sqrt{t/2} & ,t \leq 0.2 \\ 0.49-(0.11t)^{-1/2} & ,t \in (0.2,1] \\ 0.63-(0.26t)^{-1/2} & ,t \in (1,5] \\ 0.77-(0.58t)^{-1/2} & ,t \in (5,15] \\ \{2log(t)-loglog(t)-log(16\pi)\}^{1/2} & ,otherwise \end{cases}$$

approximates the boundary of the continuation region, delineating the set of iterations for which it is suboptimal to stop the exploration of potential *delegatees*.

Algorithm 2 Dynamically Indexed Delegation (DID)

```
Input: $P := \{a_i, ad_i\}_{i \in N}$: Tuples of agents and their neighbours, $s$: Array of probabilities of
    successful execution, $\delta$: Array of time-discounting parameter.
Output: $S$: Sequence of agents receiving a delegation request, $\mu$: Array of probabilities of suc-
    cessful delegation.
 1: function DID($P_i; \delta_i, s_i$)
 2:     $S \leftarrow \emptyset$, $\mu \leftarrow \{\mu_i\}_{i \in N}$, $\mu_i \sim Beta(1,1)$.
 3:     for i=1$\rightarrow N$ do
 4:         $ad_i \leftarrow \{a_j\}_{j \neq i \in K}, CountSuccess_{a_i} \leftarrow 0, \delta_i \leftarrow [0.8, 1)$
 5:         $\alpha_i = CountSuccess_{a_i}, \beta_i = CountFailure_{a_k \in ad_i}$
 6:         $\mu_i \leftarrow \frac{1}{(1+\beta_i/\alpha_i))}$
 7:         $G_{i,j} \leftarrow \mu_j + (\frac{\mu_j(1-\mu_j)}{\alpha_j+\beta_j+1})^{1/2}\psi(1/(\alpha_j+\beta_j+1)log(\delta_i^{-1}))$
 8:         $m \leftarrow argmax(\{G_{i,k}\}_{k \in ad_i})$
 9:         if $a_m \neq a_i$ then
10:             $S \leftarrow S \cup \{a_m\}$
11:             return $DID(P_m; \delta_m, s_m)$
12:         else
13:             Self-execute
14:         if $Outcome == True$ then
15:             $CountSuccess_{a_i} \leftarrow CountSuccess_{a_i} + 1$
16:         else
17:             $CountFailure_{a_i} \leftarrow CountFailure_{a_i} + 1$
18:         return $Outcome$
```

4 Evaluation

Having described our MAB and quitting game based heuristics, we now turn to evaluating their effectiveness. We begin this section by detailing our experimental setup, following which we describe our experiments and results.

4.1 Experimental Setup

Our evaluation consisted of running the various heuristics over 1000 delegation requests, each run over 100 different graphs representing different possible agent delegations. The algorithms were tested on two types of structures: 4-level directed trees (as in Figure 2), and networks of randomly formed neighbourhoods (as in Figure 1). The trees have a branching factor of 5 neighbours per node, with a final population of 156 agents. The random networks have a fixed population of 100 agents, some of which may not be reachable. Agents in random networks also possess 5 neighbours each, sampled from all available nodes excluding their immediate predecessor and the root.

We experimented with different parameters for each of the heuristics. For ϵ-greedy, ϵ takes on values between 0.05 and 0.1 [17]. Thompson Sampling was recovered from a Bayesian variation of the same algorithm with no exploration. The discount factor in DID ranged within [0.8,1), as to remain consistent with the closed-form approximation to the Gittins Index [7]. The initial probabilities of delegation were sampled from an uninformative Beta distribution.

For each heuristic, we measured the probability that a delegation would be successful after the nth iteration (averaged over the 100 runs), as well as the regret value for the action. This latter value is computed as the difference between the probability that a task would be successfully executed if the optimal delegation path was followed, and the final likelihood of successful execution.

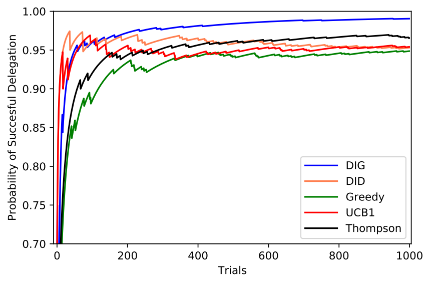
4.2 Results

Figure 4a shows the performance of the various heuristics over directed trees. We observe that the DIG heuristic significantly increases the chance of successful delegation when compared to other approaches. Thompson Sampling appears to outperform the remaining approaches, but takes longer to approach its optimal value than other techniques.

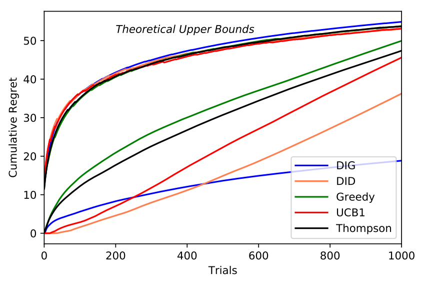
With regards to regret, we observe (Figures 4b and 5a) that DIG by maximising the likelihood of successful delegation also minimises its regret, and that this relationship holds for the remaining algorithms. Furthermore, none of the algorithms obtain levels of regret greater than UCB1's theoretical upper regret bound (Figure 4b and Table 1).

Turning to random networks, Figure 6a demonstrates that DIG and DID outperform all other approaches. It appears that the rate of convergence for Thompson sampling significantly lags behind the other approaches. Our results for regret (Figure 6b) are similar to those for directed trees.

[1] The mean rate of convergence was approximated by the error of deviating from a probability of delegating equal to 1 (e_t), over the first 175 trials i.e., $q \approx \frac{log\left(e_{t+1}/e_t\right)}{log\left(e_t/e_{t-1}\right)}, t \in \{1, \ldots, 175\}$. The cut-off point was obtained through the Welch method [19].

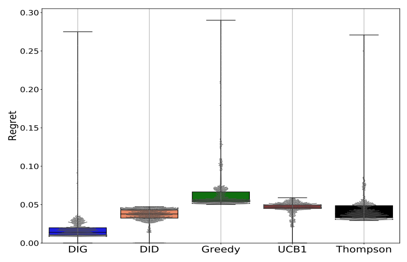


(a) Probabilities of Successful Delegation

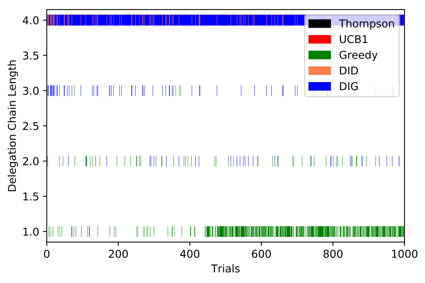


(b) Regret Metrics and Upper Bounds

Fig. 4: Comparative Performance over Directed Trees



(a) Dispersion of Regret



(b) Persistence of the Chain Length

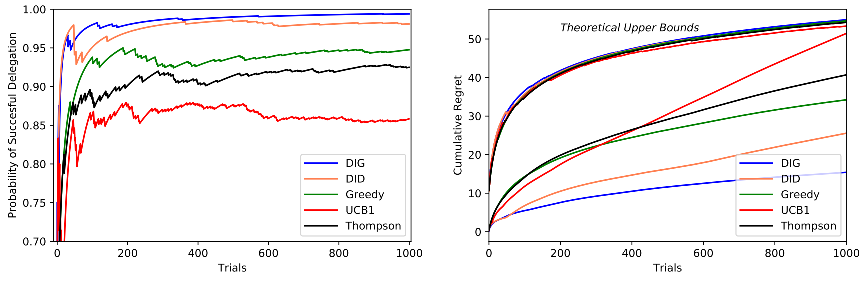
Fig. 5: Comparative Performance over Directed Trees

If we consider the length of the resulting delegation chains, we observe that in directed trees (Figure 5b) all algorithms occasionally create chains which span the height of the tree, though ϵ-greedy algorithms usually converge to a single delegation instance. We believe that the latter is due to the algorithm's focus on exploitation over exploration. In the case of random networks, this behaviour changes, with ϵ-greedy exploring the network at length, while other approaches quickly converge to different delegation chain lengths.

On account of the difference in the number of neighbours, and the presence of cycles, the variance of marginal regret is less uneven, but larger on average in the random graph case. There are more pronounced differences in the levels of regret as new agents are discovered every trial, as shown in Figure 7b. Indeed, DIG settles at a 2-agent long chain, leading to exceptional levels of successful delegation. In this sense, DIG can be considered the most efficient algorithm.

Algorithm	**Network Structure**	**Probability of Successful Delegation**	**Mean Rate of Convergence** [1]	**Mean Regret**
DIG	D.T.	**0.975**	*0.498*	**4.60**
	R.N.	**0.985**	*0.434*	**10.821**
DID	D.T.	0.958	0.363	7.766
	R.N.	0.974	0.324	15.842
ϵ-Greedy	D.T.	0.927	0.437	31.352
	R.N.	0.931	**0.608**	17.596
Thompson Sampling	D.T.	0.947	**0.731**	21.281
	R.N.	0.906	0.227	21.779
UCB1	D.T.	0.948	0.387	13.995
	R.N.	0.858	0.172	33.689

Table 1: Relative Performance over Directed Trees (D.T.) and Random Networks (R.N.)



(a) Probabilities of Successful Delegation (b) Regret Metrics and Upper Bounds

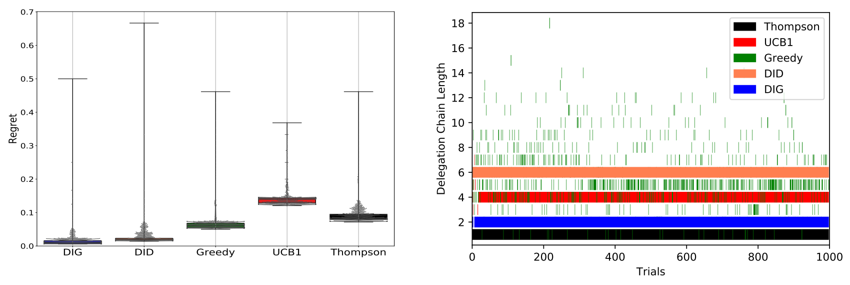
Fig. 6: Comparative Performance over Random Networks

5 Discussion and Future Work

Our results demonstrate that the DIG strategy outperforms other approaches when dealing with recursive delegation problems, and, as future work, we intend to investigate the theoretical properties of the heuristic to further understand its properties and why this result emerges.

We believe that our approaches operate better than existing heuristics due to the violation of the stationarity assumption in our domain. Our DID heuristic has similarities to the manner in which the generalised Gittins Index is computed under weaker forms of stationarity [11], suggesting the incorporation of evolutionary algorithms into future research in the domain of recursive delegation.

By construction, delegation in our MAB framework conforms to a multilevel linear program, where new delegation problems lie embedded in the constraints restricting every agent's objective. We intend to validate a similar mapping between DIG and multilevel bilinear programs against recent work on (stochastic) multilevel optimisation



(a) Dispersion of Regret (b) Persistence of the Chain Length

Fig. 7: Comparative Performance over Random Networks

problems [6], tracing back to questions on stationarity and the pertinence of evolutionary, hierarchical, and genetic algorithms [9].

Another strand of future work which we are actively pursuing involves increasing the empirical faithfulness of our approach. This means introducing resource constraints, explicit rewarding schemes, and potential costs to the delegation problem, by borrowing ideas from the principal-agent theory literature [20], and results from coalitional game theory [14].

There is little work in the computational trust community dealing with recursive delegation. To our knowledge, the only work which addresses these issues is [13] and [4]. In the former, the authors consider a supply chain problem and model it via recursive MABs, but focus on budget constraints for each arm, solving local bandit problems in parallel to identify trustworthy suppliers. [4] also consider the problem of recursive delegation, and evaluate how simple algorithms for assigning responsibility for task delegation failure across the delegation chain, affect the performance of the system.

6 Conclusions

In this paper we described the recursive delegation problem, and empirically demonstrated that a heuristic based on quitting games outperforms different multi-arm bandit based techniques, namely UCB1, ϵ-greedy, Thompson Sampling, and Lai and Brezzi's numerical approximation to the Gittins Index. Our heuristic outperforms these approaches both with regards to regret, and the probability of successful delegation over different graph topologies.

Our results are directly applicable to multi-agent system marketplaces, and address an oft-ignored issue in computational trust research, which usually only considers non-recursive task delegation. In this regard, extensions to include explicit rewarding schemes and resource constrains seem a fruitful direction of future research. We believe they will give rise to decisive contributions to computational trust theory and AI, if further pursued along the lines of hierarchical reinforcement learning in non-stationary environments [12, 18].

References

1. Agrawal, S., Goyal, N.: Analysis of thompson sampling for the multi-armed bandit problem. In: Conference on Learning Theory. pp. 39–1 (2012)
2. Auer, P., Fischer, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning **47**, 235–256 (2002)
3. Brezzi, M., Lai, T.L.: Optimal learning and experimentation in bandit problems. Journal of Economic Dynamics and Control **27**(1), 87–108 (2002)
4. Burnett, C., Oren, N.: Sub-delegation and trust. In: AAMAS. pp. 1359–1360. IFAAMAS (2012)
5. Chapelle, O., Li, L.: An empirical evaluation of thompson sampling. In: Advances in neural information processing systems. pp. 2249–2257 (2011)
6. Franke, S., Mehlitz, P., Pilecka, M.: Optimality conditions for the simple convex bilevel programming problem in banach spaces. Optimization **67**(2), 237–268 (2018)
7. Gittins, J., Glazebrook, K., Weber, R.: Multi-armed bandit allocation indices. John Wiley & Sons (2011)
8. Gutin, E., Farias, V.: Optimistic gittins indices. In: Advances in Neural Information Processing Systems. pp. 3153–3161 (2016)
9. He, X., Zhou, Y., Chen, Z.: Evolutionary bilevel optimization based on covariance matrix adaptation. IEEE Transactions on Evolutionary Computation (2018)
10. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American statistical association **58**(301), 13–30 (1963)
11. Koulouriotis, D.E., Xanthopoulos, A.: Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems. Applied Mathematics and Computation **196**(2), 913–922 (2008)
12. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In: Advances in neural information processing systems. pp. 3675–3683 (2016)
13. Sen, S., Ridgway, A., Ripley, M.: Adaptive budgeted bandit algorithms for trust development in a supply-chain. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 137–144. AAMAS '15, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2015), http://dl.acm.org/citation.cfm?id=2772879.2772900
14. Skibski, O., Michalak, T.P., Rahwan, T., Wooldridge, M.: Algorithms for the shapley and myerson values in graph-restricted games. In: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems. pp. 197–204. International Foundation for Autonomous Agents and Multiagent Systems (2014)
15. Solan, E., Vieille, N.: Quitting games. Mathematics of Operations Research **26**(2), 265–285 (2001)
16. Solan, E., Vieille, N.: Quitting games–an example. International Journal of Game Theory **31**(3), 365–381 (2003)
17. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. Cambridge, MA: MIT Press (2011)
18. Vezhnevets, A.S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., Kavukcuoglu, K.: Feudal networks for hierarchical reinforcement learning. arXiv preprint arXiv:1703.01161 (2017)
19. Welch, P.D.: The statistical analysis of simulation results. The computer performance modeling handbook **22**, 268–328 (1983)
20. Zhang, H., Zenios, S.: A dynamic principal-agent model with hidden information: Sequential optimality through truthful state revelation. Operations Research **56**(3), 681–696 (2008)