Adaptive Budgeted Bandit Algorithms for Trust Development in a Supply-Chain

Sandip Sen
Tandy School of Computer Science
The University of Tulsa
sandip@utulsa.edu

Anton Ridgway
Tandy School of Computer Science
The University of Tulsa
anton-ridgway@utulsa.edu

Michael Ripley
Tandy School of Computer Science
The University of Tulsa
michael-ripley@utulsa.edu

ABSTRACT

Recently, an AAMAS Challenges & Visions paper identified several key components of a comprehensive trust management has been understudied by the research community [13]. We believe that we can build on recent advances in closely related research in other sub-fields of AI and multiagent systems to address some of these issues. For example, the budgeted multi-armed bandit problem involves pulling multiple arms with stochastic rewards with the goal of maximizing the total reward generated from those arms, while keeping the cost of pulling the arms beneath a given budget. We argue that multi-armed bandit algorithms can be adapted to address research issues in trust engagement and evaluation components of a comprehensive trust management approach. To support this proposition, we consider a supply-chain application, where a tree of dependent supplier agents can be considered as an arm of the online bandit problem with budget constraints. Each of the nodes in the supply chain must then solve their local bandit problem in parallel to determine which of its sub-suppliers is most trustworthy. We use new arm-selection strategies, and demonstrate how they can be gainfully applied to the trust-based decision-making in the supply chain to reduce time to production and hence improve utility by timely delivery of products.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Performance, Reliability, Algorithms

Keywords

exploration-exploitation; trust management; contracting

1. INTRODUCTION

Research on trust in multiagent systems has given us a variety of conceptual frameworks to view trust and effective algorithms for evaluating the trustworthiness of other agents given the history of mutual interactions [7, 9, 11, 16, 17]. However, it has been posited that to fully harness the effectiveness of trust based decision making, it is critical to develop a more comprehensive approach to trust management that addresses not only trust evaluation but also provides strategic reasoning procedures that determine who to interact with, in what context, and how to best utilize the resultant knowledge about the trustworthiness of others [13]. The goal then is to develop proactive trust mechanisms that explore possible fruitful partnerships, create situations where interactions will provide discriminatory evidence of the trustworthiness of potential partners, and use a well-thought-out plan of how to best utilize the trusted partners given expectations of future goals, plans and resource requirements.

In this paper, we consider the problem of proactive engagement of potential long-term partners to determine who can be trusted for providing consistent service on which an agent's goal is critically dependent. Whereas most of the research on trust in multiagent systems focuses on after-the-event, offline evaluation of interaction histories to determine trustworthiness of partners, real-world scenarios demand forward-looking, online schemes that has to choose to engage interaction partners where the process of engagement incurs a cost. This "exploration cost" must be balanced against possible gain or "exploitation benefit" in the long run from the knowledge of who the more trustworthy partners are. Research in machine learning, and in particular, reinforcement learning has long confronted similar exploration versus exploitation dilemmas [12]. However, most of these research has ignored the time and cost of exploration, and focused primarily on proving convergence to optimal policies for solving the underlying Markov Decision processes in the limit, i.e., without time or budget constraints. In particular, the multi-armed bandit (MAB) problem has been proposed as a theoretical framework for evaluating the utility of interacting with multiple entities with stochastically varying performance [2, 1].

Though this approach would appear to be a natural mapping of the problem of evaluation of trust in new partners, on closer inspection a key missing aspect underlines the fundamental difference between the two. The basic MAB model does not consider any cost for pulling an arm, whereas interacting with a partner to gather further information to evaluate their trustworthiness involves risks, time and resource commitment, as well as other costs. More recently, however, researchers have considered augmentations of the basic MAB problem, that consider the cost of pulling arms in combination with a budget limit for exploration [8, 14, 15], i.e., a resource constraint that necessitates strategic engagement with potential partners to quickly identify partnerships which are likely to deliver maximal long-term benefits. We believe that solution approaches for the Fixed-Cost Multi-Armed Bandit Problem with Budget Constraints (MAB-BF) can be adapted to address the issue of strategic engagement and evaluation of potential partner's

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

trustworthiness under a number of settings and trust metrics such as reliability, fairness, quality of performance, timeliness, etc. The MAB-BF has already been utilized for a wide range of reinforcement learning applications, including bidding in ad exchanges, bid optimization in search, and service provider selection in cloud computing [3, 5, 4, 6, 10].

Various metrics for determining the trustworthiness of a partner has been proposed. In this paper, we consider performance, in terms of reliability, as the objective criteria for trustworthiness. This choice is motivated by the following observation [13]: *Trust in another agent reduces the uncertainty over that agent's independent actions which positively correlates with the truster's utility.* We consider existing and recently developed approaches to addressing the MAB problem with cost and budget constraints to proactively engage and evaluate potential partners to gauge their reliability and hence long-term trustworthiness. To evaluate the efficacy of these approaches, we introduce a Supply Chain domain where a manufacturer has to procure raw materials from contractors of initially unknown reliability. Contractors in turn have to depend on sub-contractors to produce their deliverables. The goal of each agent in the supply chain is to reduce their time to delivery of their product for which they have to strategically engage their sub-contractors to discriminate sub-contractors of differing reliability.

We introduce three new algorithms for addressing the MAB-BF problem. First, we formally present the MAB-BF problem (Section 2); then, we introduce recent algorithms proposed for this variant of the MAB problem, together with our own proposed algorithms, highlighting their interrelationships and differences (Section 3). We then introduce the supply chain domain used for evaluating the effectiveness of the proposed algorithms in determining the trustworthiness of the sub-contractors under varying budget constraints and performance variability (Section 4). In Section 5 we describe the simulation framework used for experiments, and in Section 6 present a series of experimental results from a variety of scenarios , varying the branching factor of the arms in each tree, the payoff distributions of the arms, and the budget available to sample from the arms. In conclusion, in Section 7 we summarize the key findings from the comparative experimental evaluation and present thoughts for future research.

2. FIXED-COST MULTI-ARMED BANDIT WITH BUDGET CONSTRAINT

We now formally introduce the version of MAB-BF problem where the goal of the agent is to maximize the reward obtained by pulling a set of $\mathcal{A} = \{1, \ldots, K\}$ arms, the ith arm has a fixed cost c_i per pull, and arm rewards are drawn from an unknown distribution with a mean value of μ_i. The agent has at its disposal a total budget of B. Let $C = \langle c_1, \ldots, c_K \rangle$ and $\mu = \langle \mu_1, \ldots, \mu_K \rangle$ refer to the vector of pulling costs and mean rewards for the K arms. We refer to the above problem as the Fixed-Cost Multi-Armed Bandit with Budget Constraint (MAB-BF), defined by the tuple $P = (\mathcal{A}, B, C, \mu)$. The goal of an agent facing such a problem is to choose a sequence of arms that optimizes the expected reward without exceeding the total budget for arm pulls. Because the number of arm pulls in a given sequence is limited by the costs of the particular arms chosen, we generally consider the utility ratio $\frac{\mu_i}{c_i}$ rather than μ_i alone to determine the priority of a given arm i. It should be noted that while one could improve overall performance somewhat by incorporating the standard deviation of reward received, or any other statistical measures, together with the utility ratio to produce a more general utility index, the result would only be a general optimization, applicable to any of the algorithms we consider; for simplicity's sake, we choose to focus on the utility ratio only, and leave the determination of a best-case utility index for another setting. Finally, note that the μ values are not available to the agent.

The number of arm pulls that an agent can make is dependent on the multiset of arms it decides to pull. We now introduce some notations to refer to sequences of arm pulls:

- $S = \langle a_1, \ldots, a_t \rangle$ refers to the sequence of arms pulled by the agent in the first t attempts. We will use $|S|$ to refer to the number of arm pulls in the sequence S and $S(i), \forall i \in \{1, \ldots, |S|\}$ to refer to the ith arm pulled in that sequence. Also we denote the subsequence of pulls in S from $t = t_1, \ldots, t_2$ as S_{t_1,t_2}. We will use $a \in S$ to test for the existence of an arm a in the sequence S or to range over the set of arms in the sequence.
- n_a^S represents the number of times arm $a \in \mathcal{A}$ was pulled in sequence S.
- A_t is the set of arms which can still be pulled after the sequence of arm pulls $S_{1,t}$ with the remaining budget $B_t = B - \sum_{i=1}^{t} c_{S(i)}$, i.e., $A_t = \{a | a \in \mathcal{A} \wedge c_a \leq B_t\}$.
- A sequence S is *valid* if it does not violate the budget constraint, i.e., $\sum_{a \in S} c_a \leq B$.
- $\mathcal{S} = \{S | S \text{ is valid}\}$ is the set of all valid sequences. For the rest of the paper we will use the term 'sequence' for valid sequences.
- $G_S = \sum_{i}^{|S|} r_{a_i,i}$ is the total reward obtained from a valid sequence S, where $r_{a_i,i}$ is the reward returned from arm $a_i \in \mathcal{A}$ pulled at time i. As the rewards generated are non-deterministic, we are more interested the sum of the *mean* rewards in the sequence; that is, the *Expected Total Reward of sequence* S, $E[G_S] = \sum_{i}^{|S|} \mu_{a_i}$. Correspondingly, we are interested in the optimal sequence S^* if the reward distributions for the arms were known,

$$S^* = \arg\max_{S \in \mathcal{S}} E[G_S] \tag{1}$$

  and the associated payoff $E[G_{S^*}]$. Note that since the mean rewards for the arms are not known *a priori*, we will only know G_S.
- Given a sequence of arm pulls, S, an estimate of the mean reward for each of the sampled arms, μ_S, can be formed. Then $\mu_S = \langle \mu_1^S, \ldots, \mu_K^S \rangle$ where $\mu_i^S, \forall i \in \mathcal{A}$, is the average reward obtained from the n_i^S pulls of arm i in the sequence S:

$$\mu_i^S = \frac{1}{n_i^S} \sum_{j=1}^{|S|} I(S(j) = i) r_{i,j} \tag{2}$$

  where $I(\cdot)$ is the Indicator function.
- The expected regret, $\mathcal{R}(S)$ of the sequence S is then calculated as the difference: $\mathcal{R}(S) = E[G_{S^*}] - E[G_S]$. The desirability of a sequence of arm pulls can be measured by its expected regret and the problem of optimizing expected reward can be mapped into the problem of minimizing expected regret.

3. MAB-BF ALGORITHMS

In this section we introduce the different algorithms we experiment with. We first introduce some existing algorithms and then discuss our proposed approaches.

3.1 Existing Algorithms

We now describe some existing algorithms for the MAB problem; the first two (ϵ-first, Greedy) have an initial exploration phase for estimating the mean rewards of the K arms, and in the subsequent exploitation phase pull the arms in a greedy manner, and the last (fKDE) one stochastically integrates exploration and exploitation while narrowing its focus:

Budget-Limited ϵ-first: This algorithm, henceforth referred to as ϵ-first, uniformly selects from the set of arms, performing unordered sweeps of each before beginning again, until its exploration budget, ϵB, is exhausted (not enough budget remains to pull even the minimum cost arm[1]). Let $S^{explore}_{\epsilon-\text{first}}(P)$[2] be the sequence of arm pulls for the exploration phase for an ϵ-first approach with MAB-BF problem P. At the end of the exploration phase, an estimate of the mean rewards, $\mu_{S^{explore}_{\epsilon-\text{first}}}$ is calculated and subsequently used in the exploitation phase. Next, the arms are sorted by the ratio of their estimated mean to pulling cost (their reward-cost ratio). The best such arm, $I^{max} = \arg\max_{i \in \mathcal{A}} \frac{\mu_i^{S^{explore}_{\epsilon-\text{first}}}}{c_i}$, which we refer to as the *active arm*, is pulled until not enough of the exploitation budget, $(1-\epsilon)B$, is left for one more pull of that arm. This process is repeated with the rest of the arms until the budget is completely exhausted (not enough remains to pull even the lowest cost arm). We refer to the sequence of arms pulled in this exploration phase as $S^{exploit}_{\epsilon-\text{first}}(P)$ and the combined exploration-exploitation sequence as $S_{\epsilon-\text{first}} = S^{explore}_{\epsilon-\text{first}}(P) + S^{exploit}_{\epsilon-\text{first}}(P)$. This algorithm is presented in Algorithm 1.

One key difference between the original formulation of this algorithm[14] and our implementation of it is that our version continues to update the estimates of the arms during the exploitation phase, so that the active arm in that phase can change if its utility ratio drops below that of another arm, even if its cost is still affordable. We refer to our version of algorithms that behave this way as *online-exploitation* variants, in contrast to the original *offline-exploitation* variants. Experiments have shown that the online variants significantly outperform the offline variants in most scenarios and hence in this paper we present results with the online variants only, unless otherwise noted.

Greedy Algorithm: The Greedy algorithm is a special case of the ϵ-first algorithm where the exploration budget only allows each arm to be pulled once before the exploitation phase begins. This algorithm is remarkable because to our knowledge it has not been used before for the MAB-BF problem, but its online-exploration variant nevertheless performs exceptionally well on average. Note that since we require all arms to be pulled at least once, when $\epsilon \leq \frac{\sum_{i=1}^{K} c_i}{B}$, ϵ-first algorithms are reduced to the Greedy algorithm.

Fractional Knapsack-based Decreasing ϵ-greedy (fKDE): The fractional KDE algorithm [14] is a decreasing exploration over time approach which first pulls the arms uniformly γ times and thereafter pulls the arm with the highest estimated reward-cost ratio with increasing probability. The probability of uniform random exploration after t arm pulls is set to $\varepsilon_t = \min(1, \frac{\gamma}{t})$. Otherwise, the arm with the highest estimated reward-cost ratio, based on the sequence of arms pulled, is chosen.

Input : $P = (\mathcal{A}, B, C, \mu)$: MAB-BF problem;
ϵ, exploration fraction of budget
Output: S, a sequence of arm pulls;
μ', a vector of estimated mean rewards from the K arms;
G, total reward from all arm pulls;

Exploration phase:
$t \leftarrow 1$; $S \leftarrow \varnothing$; $G \leftarrow 0$; ExplorationRounds $\leftarrow \left\lfloor \frac{\epsilon B}{\sum_{i=1}^{K} c_i} \right\rfloor$;
for $i = 1 \rightarrow K$ **do**
  $\mu'_i \leftarrow 0$
for $n = 1 \rightarrow$ *ExplorationRounds* **do**
  for $i = 1 \rightarrow K$ **do**
    pull arm i to obtain $r_{i,t}$; $G \leftarrow G + r_{i,t}$;
    $S \leftarrow S + i$; // Add i to sequence S
    $\mu'_i \leftarrow \mu'_i + r_{i,t}$;
    $t \leftarrow t + 1$;
for $i = 1 \rightarrow K$ **do**
  $\mu'_i = \frac{\mu'_i}{n_i^S}$
; // form reward mean estimates
Exploitation phase:
RemainingBudget = $B -$ ExplorationRounds $* \sum_{i=1}^{K} c_i$;
$A' = \mathcal{A}$; // Initialize available arms
while *RemainingBudget*$\geq \min_{i \in \mathcal{A}'} c_i$ **do**
  $I^{max} = \arg\max_{i \in A} \frac{\mu_i}{c_i}$; // pick best arm
  if *RemainingBudget*$\geq c_{I^{max}}$ **then** // if budget allows to pull arm
    pull arm I^{max} to get reward $r_{I^{max},t}$;
    $G \leftarrow G + r_{I^{max},t}$;
    $\mu'_{I^{max}} \leftarrow \frac{\mu'_{I^{max}} * n^S_{I^{max}} + r_{I^{max},t}}{n^S_{I^{max}} + 1}$; // update mean reward estimate of arm
    $S \leftarrow S + I^{max}$;
    RemainingBudget $\leftarrow$ RemainingBudget$-c_{I^{max}}$;
    $t \leftarrow t + 1$;
  else
    $A' \leftarrow A' \setminus \{I^{max}\}$; // eliminate arm
return (S, μ', G);

Algorithm 1: The ϵ-first algorithm.

3.2 New Algorithms

Now, we present algorithms that we have developed for the MAB-BF problem. We believe these algorithms explore more intelligently compared to their predecessors; the choice of arms to be pulled during exploration is driven by either the number of arms $|\mathcal{A}|$, or the distribution of the arm rewards in μ and the costs in C; past algorithms only made use of uniform or minimal exploration phases, and fixed the exploration budget without consideration of the bandit at hand. Also note that all the algorithms that we introduce are *online* algorithms, i.e., an eliminated arm may be reconsidered if the reward-cost ratio of a sufficient number of previously preferred arms drops below the corresponding ratio of this arm upon further sampling.

l-split (lS): This is a generalized Greedy approach. Instead of eliminating all but one arm after the first pass, the `lS` algorithm successively eliminates $(1 - \frac{1}{l})$ of the arms after each pass: if $A_{lS}(p)$ is the number of surviving arms after p splitting passes of the algorithm, then $A_{lS}(p+1) = \lceil \frac{1}{l} A_{lS}(p) \rceil$. After $\lceil \log_l K \rceil$ passes, `lS` narrows down the choice to one, and thereafter performs greedy exploitation. The algorithm

[1] We only consider scenarios where the exploration budget allows us to pull every arm the same number of times in the exploration phase.

[2] We will omit the problem argument P in cases where the context is clear.

is presented in detail in Algorithm 2. The simplest of this family of algorithms is the 2-split (2S) or halving algorithm, which successively eliminates approximately half of the underperforming arms after each pass.

Progressive exploration ϵ-first (PEEF): The PEEF algorithm was developed upon a careful evaluation of the ϵ-first algorithm. Whereas the latter expends its exploration budget uniformly over the set of K arms, we conjectured that in a number of scenarios, particularly those with a large number of arms, there might be some low return arms that can be quickly discarded. More importantly, we believe that the exploration budget can be better utilized to tease out differences between similar, high reward-cost ratio arms by visiting them with increasing frequency. Hence, rather than uniform exploration, we propose a progressive exploration scheme where we perform a $l-$split operation, as in the lS algorithm after each pass. The difference with that algorithm is that in PEEF the splitting value l is calculated such that the number of remaining arm choices is reduced to 1 approximately at the end of the exploration phase. Given an exploration budget of ϵB then we want l to be such that the following condition holds:

$$\sum_{j=1}^{\log_l K} \frac{K}{j} c_{avg} = \epsilon B, \tag{3}$$

where $c_{avg} = \frac{1}{K}\sum_{i=1}^{K} c_i$, is the average cost of pulling an arm[3]. Solving this equation for l we obtain $l = \frac{\epsilon B - 1}{\epsilon B - K}$. For obvious reasons, we will perform a pairwise comparison of the ϵ-first algorithm and the corresponding PEEF algorithm for different scenarios in the experimental section.

Survival of the Above Average (SOAAv): This algorithm also successively narrows down the set of active arms by eliminating underperforming arms. But rather than eliminating a fixed number of arms after each pass, it eliminates arms whose estimated reward-cost ratio is below $(1+x)$ times the average of such ratios of the arms in the last pass.

Setting $x = 0$ means only above average individuals survive from one pass of the arms to the next. Note again that this is an online-exploration approach where a previously eliminated arm can come back into the active set if estimates of other active arms drop. This algorithm is presented in more detail in Algorithm 3.

Of the above algorithms, both lS and PEEF are rank-based algorithms, where arms are eliminated based on their ranking by reward-ratio cost ratios, whereas only the SOAAv algorithm is a *value-based* approach, where arms with estimated reward-cost ratios below a certain factor of the average of the currently active set are eliminated.

Input : $P = (\mathcal{A}, B, C, \mu)$: MAB-BF problem;
l, elimination factor
Output: S, a sequence of arm pulls;
μ', a vector of the arms' sample mean rewards;
G, total reward from all arm pulls;

$t \leftarrow 1$; $S \leftarrow \varnothing$; $G \leftarrow 0$; RemainingBudget $\leftarrow B$; NumPasses $\leftarrow 0$
for $i = 1 \rightarrow K$ **do**
  $\mu'_i \leftarrow 0$
$A' = \mathcal{A}$; // Initialize available arms
while $A' \neq \varnothing$ **do**
  foreach $a \in A'$ **do**
    if *RemainingBudget*$\geq c_{a_i}$ **then** // if budget allows to pull arm
      pull arm a to obtain reward $r_{a,t}$;
      $G \leftarrow G + r_{a,t}$;
      $\mu'_a \leftarrow \frac{\mu'_a * n_a^S + r_{a,t}}{n_a^S + 1}$; // update mean reward estimate of arm
      $S \leftarrow S + a$;
      RemainingBudget $\leftarrow$ RemainingBudget$-c_a$;
      $t \leftarrow t + 1$;
  $NumPasses \leftarrow NumPasses + 1$;
  $A' = \varnothing$;
  NumToPull $\leftarrow \lceil \frac{K}{l^{NumPasses}} \rceil$;
  while *NumToPull* > 0 *and* $A - A' \neq \varnothing$ **do**
    $A' \leftarrow A' \cup \{\arg\max_{i \in A} \frac{\mu_i}{c_i} \mid c_{a_i} \leq RemainingBudget, A_i \notin S'\}$;
    NumToPull $\leftarrow$ NumToPull - 1;
return (S, μ', G);

Algorithm 2: The $l-$Split algorithm.

4. SUPPLY-CHAIN MODEL

We now introduce the supply-chain model where contractors have to engage strategically with sub-contractors of initially-unknown reliability (trustworthiness). Each interaction has a cost and the contractor's goal is to distinguish the trustworthiness of all of its sub-contractors given a fixed budget to pay for the interactions. This domain allows us to examine how the algorithms developed for the budget-constrained MAB problem perform in complex, large-scale systems, while simultaneously examining the algorithms' efficiency in dealing with trust-based decision making. The trust that the agent places in each of its sub-contractors in the chain is based on its past observations while engagement decisions have to be predicated on remaining uncertainties about discriminating between potential partners.

In our supply-chain model, there is a root agent, which has access to a fixed set of contractor agents, represented as the set of arms $\mathcal{A}_{0,0} = \langle a_{1,0}, \ldots, a_{1,k} \rangle$, where k is the branching factor. Each of the other agents $a_{l,i}$, in turn, has access to its own set of sub-contractors $\mathcal{A}_{l,i} = \langle a_{l+1,0}, \ldots, a_{l+1,k} \rangle$, where l is the current level in the tree. This gives rise to a supply-chain tree, with the original agent $a_{0,0}$ at the root. Each agent besides the root agent possesses a fixed contracting cost $c_{l,i}$. The utility that each agent returns to its contracting agent is the inverse of the time that it takes to complete the task. This time is determined by the time returned by the agent's own sub-contractors, added to a value drawn from a Gaussian distribution internal to the agent, with mean $\mu_{l,i}$ and standard deviation $\sigma_{l,i}$; leaf agents' times are determined by their own distributions only. For our implementation of this model, we chose to hold the branching factor fixed for all the levels of each branch. Additionally, to maintain a reasonable level of variation in our experiments, we chose to give each agent in $\mathcal{A}_1$ identical sub-trees.

Especially important is that, because each algorithm must perform its own bandit problem on its sub-contractors simultaneously, and because no agent has the opportunity to sample arms when it is not pulled, most agents will not be able to complete their explo-

[3]Note that this calculation of l is necessarily approximate, both for the use of c_{avg} as the cost per unit pull during the exploration phase and in the use of $\log_l K$ as the number of passes during the exploration phase.

Input : $P = (\mathcal{A}, B, C, \mu)$: MAB-BF problem;
x, elimination factor
Output: S, a sequence of arm pulls;
μ', a vector of the arms' sample mean rewards;
G, total reward from all arm pulls;

$t \leftarrow 1$; $S \leftarrow \varnothing$; $G \leftarrow 0$; RemainingBudget $\leftarrow B$;
for $i = 1 \rightarrow K$ **do**
  $\mu'_i \leftarrow 0$
$A' = \mathcal{A}$; `// Initialize available arms`
while *RemainingBudget* $\geq \min_{i \in A'} c_i$ **do**
  numPullsInPass=0; passAverageRatio=0;
  foreach $a \in A'$ **do**
    if *RemainingBudget* $\geq c_a$ **then** `// if budget allows to pull arm`
      pull arm a to obtain reward $r_{a,t}$;
      $G \leftarrow G + r_{a,t}$;
      $\mu'_a \leftarrow \frac{\mu'_a * n_a^S + r_{a,t}}{n_a^S + 1}$; `// update mean reward estimate of arm`
      $S \leftarrow S + a$;
      RemainingBudget $\leftarrow$ RemainingBudget$-c_a$;
      $t \leftarrow t + 1$;
      passAverageRatio $\leftarrow$ passAverageRatio+$\frac{r_{a,t}}{c_a}$;
      numPullsInPass$\leftarrow$ numPullsInPass+1;
    else
      $A' \leftarrow A' \setminus \{a\}$; `// eliminate arm`
  if *numPullsInPass >0* **then**
    passAverageRatio $\leftarrow \frac{\text{passAverageRatio}}{\text{numPullsInPass}}$;
    $A' = \varnothing$
    foreach $a \in A$ **do**
      if $c_a <$*RemainingBudget &* $\mu'_a \geq (1 + x)$*passAverageRatio* **then**
        $A' \leftarrow A' \cup \{a\}$;
return (S, μ', G);

Algorithm 3: The `SOAAv` algorithm.

ration before the root agent has completed its budget. To remedy this, we set the budget of agent i on level l to be $B_{l,i,t} = kc_{l,i}$, if it is selected at time t. Thus, if an agent's budget is distributed evenly among all its sub-contractor agents, the budget of each sub-contractor will be equal to that of the original contractor, giving each agent the chance to explore its options and reach exploitation in time for its contractor to benefit. Though we chose for simplicity to fix the budget multiplier at a fairly neutral level (the parent's branching factor), this value could be altered per-agent or over time to introduce additional variability in the model. Additionally, because each agent in the tree is exploring simultaneously, we expected less aggressive bandit algorithms to have a slight advantage in this setting, since they will allow the sub-contractors more time to finish exploring before they begin exploitation.

It is also of note that, because of this set-up, no agent except for the root knows what its total budget will be until it has already spent it. Thus, for algorithms that fix their exploration budget in advance, we provided an estimated exploration budget of the same size as its contracting agent– since when budget is expended uniformly, each agent receives the same budget as its contracting agent, this seemed to be a reasonable measure that would allow these algorithms to be executed throughout the tree.

5. SIMULATION FRAMEWORK

In our simulation framework, we implemented fixed supply-chain trees where the ith agent in level l, agent $a_{l,i}$, was provided with a Gaussian reward distribution defined by $\{\mu_{l,i}, \sigma_{l,i}\}$ and a fixed engagement or contracting cost of $c_{l,i}$. The algorithms' performance in each scenario was averaged over one thousand trials. As results in experiments where arm costs were selected randomly close to 1 were qualitatively similar to those where all arm costs were 1, we use the results from the latter scenario for uniformity in the results presented here. This fixed cost constraint could easily be lifted, and does not substantially affect our results.

To evaluate the algorithms under consideration for the supply-chain problem, and to help discover which situations each could function best in, we made use of the following test scenarios. There are two sets of experiments that use different allotment of MAB-BF algorithms to the different nodes in the supply-chain:

Root-Variation Trees: In the root-variation case, the root agent employs the algorithm under consideration, while each other contractor in the tree employs the same "basic" algorithm (we chose `lS`) in every case; because the sub-tree of contractors would perform the same for each algorithm tested, we use this case for comparing all MAB-BF algorithms.

Homogeneous Trees: In the homogeneous case, we employ the same algorithm at each level of the tree. We expected that this arrangement would help to amplify each algorithm's relative strengths and weaknesses. Because some algorithms require the total budget to be known initially, but no agent except for the root knows this in advance, estimated exploration budgets are needed for these to be allowed in the homogeneous configuration. So experiments with Homogeneous trees are run only for algorithms that does not require the knowledge of the total budget.

In addition, we use different function distributions to generate the performances of the contractors in the supply-chain. For each of the root-variation and homogeneous cases, therefore, we also experimented with several node performance distributions.

Curve Distributions: The simplest function type that we used is the linear *curve-based* distribution. That is, the values in the μ_l vector were chosen so that successive $\mu_{l,i}$ values followed some linear, superlinear (concave down), or sublinear (concave up) curve, with the average time for the first arm being $\mu_{l,1} = \frac{\mu_{l,max}}{k}$ and that of the last arm being $\mu_{l,k} = \mu_l^{max}$, where μ_l^{max} is the maximum mean time for any arm on level l of the tree. The standard deviation of processing time was held the same for all agents. Thus, in the linear set, arms' means were evenly distanced. In contrast, there were more (less) arms with high reward in the superlinear (sublinear) curve distributions compared to arms with low rewards. These configurations allowed us to examine the algorithms' performance in conditions where either more (superlinear) or less (sublinear) exploration would be beneficial.

Terraced Agents: We also considered "terraced" agent organizations, where for agents in the lth level, $a_{l,i}$, a single agent with a best μ was placed together with some percentage of "good" performers (around 80% μ values compared to the best), and the rest were "poor" performers (around 20%); the payoff variance was chosen to be large enough such that the best arm was difficult to identify. Contractors with such variable performance allowed us to better examine and evaluate how intelligently algorithms expended their exploration budget; the poor contractors were chosen to penalize algorithms that explore less intelligently, while the presence of the "good" contractors required focused sampling to discern which was the best– a single sweep or any myopic choice was unlikely to be sufficient.

6. EXPERIMENTAL RESULTS

In this section, we discuss the algorithms' performance for our trust-based supply-chain scenario. To evaluate the algorithms, we focused on the reward that they generated as the initial budget varied, and as the standard deviation of completion time of each contractor (fixed at the same value for every agent) varied. The parameters values chosen for these scenario were varied to determine when each algorithm would perform at its best, and in which situations it is less effective.

In general, our experimentation showed that at low budgets, the more aggressive algorithms, that performed the least exploration, tended to perform the best. The Greedy and PEEF algorithms converged with sufficiently small budget (they were not allowed to explore any less than the Greedy, since then some of the arms would go unexplored); lS and SOAAv were a little worse, while high-exploration ϵ-first and KDE were significantly worse (see Figure 1). As it turns out, the natural advantage that the model provides to less aggressive algorithms is not enough to overcome the advantage that more decisive algorithms accrue in a budget-limited environment.

As the budget increased, the lS and SOAAv algorithms quickly outperformed the other algorithms in nearly every case. Perhaps surprisingly, the low-exploration ϵ-first method also proved very strong (nearly on-par with PEEF), while KDE, despite its superior, asymptotically-optimal regret bounds [14], performed relatively worse for small budgets. By examining individual contractor-selection sequences, we determined that this was because, in many cases, the amount of exploration performed by KDE was not justified by the savings accrued in being able to locate the best contractor. By contrast, the ϵ-first method was still often able to determine the best contractor eventually, by reordering its contractor ranking after some number of mistakes during exploitation. With a sufficiently high budget, KDE was able to overtake the low-exploration ϵ-first method (see Figure 1), since the reward it missed from increased exploration was eventually surpassed by the savings it incurred over the Greedy algorithm's method, which accrued some regret over time on average, from the trials where it could not correctly identify the best contractor.

We also saw that while the PEEF algorithm performed well with small budget (it becomes identical to Greedy when budget is sufficiently small, since the algorithm must select each contractor at least once). Its average time to completion, like Greedy, drops off relative to the others once it begins spending more time exploring– that is, the other algorithms are able to explore more intelligently. As was expected, the PEEF method follows the general pattern of the equivalent ϵ-first algorithm, while their intelligent choice of contractors allows it to remain more competitive. The fact that the ϵ-first algorithms fall behind KDE in our experiments supports our understanding that these algorithms have asymptotically worse regret bounds than KDE.

Moreover, the juxtaposition of lS and PEEF emphasize the importance of how the algorithms are parameterized. The two algorithms are structurally identical, but because PEEF's exploration budget is dependent on the size of the budget, while lS's is dependent on the number of arms, lS is better able to cope with a variety of different datasets without altering its parameters, when the given budget is of a moderate to large size. For the same reason, SOAAv's parameter is sufficiently general that the algorithm remains effectively adaptive without fine tuning.

6.1 Curve-Based Distributions for Base-Variation and Homogeneous Trees

When testing with the curve-based distributions, we consider cases where the $\sigma_{l,i}$ values were set at 20, and $\mu_{l,i}$ values were

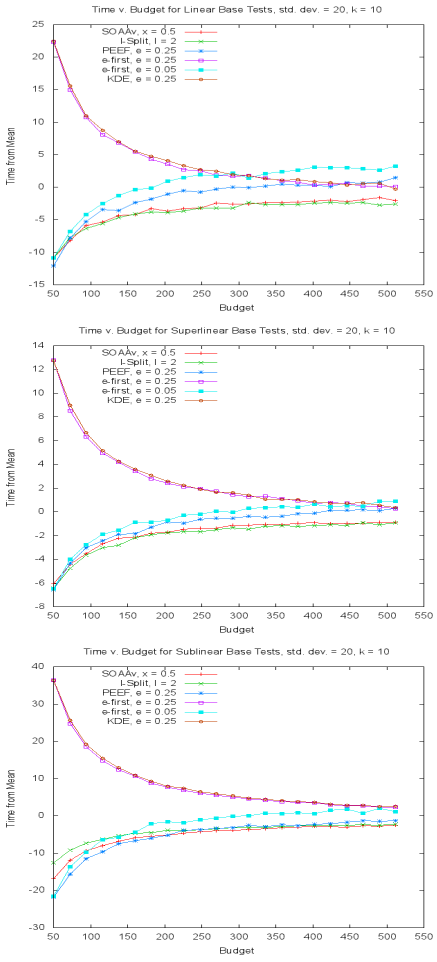

Figure 1: Time v. Budget for Curve-based Base-variation Trees - Linear (left), Superlinear (center), and Sublinear (right)

chosen 10 units apart. The branching factor was $k = 10$. All contracting costs were set to 1 unit, and contracting budgets ranged from 50 (five times the cost to pull all arms available to the root) to 500 (fifty times). Results from these experiments are plotted in Figures 1 and 2; here, time values are considered relative to the mean time values of all the algorithms to highlight differences between their performances. We can see that as budget increases, the differences between the KDE and ϵ-first algorithms become much smaller, while the other algorithms tend to spread out. The overall trends in the algorithms' rankings at low and high budgets matched those discussed earlier: the lS algorithm performed the best at large budgets, with SOAAv generally very close behind. While both PEEF and low-exploration ϵ-first were preferable to these for very small budgets, they became worse as budget became larger.

Also of note is the fact that KDE and ϵ-first are much less competitive to begin with, but begin to increase fast enough to overtake the other algorithms when budget becomes sufficiently large, as it does for ϵ-first and PEEF in the range of our experiments. This suggests that they could perform very well in very long-term cases. For the budget-limited scenario, however, this is of limited value.

In considering the sublinear and superlinear utility distribution trials, we found that the Greedy and PEEF algorithms tend to do relatively better in the sublinear case, and worse in the superlinear and linear cases. This makes sense, since these are more aggres-

Figure 2: Time v. Standard Dev for Linear, Homogeneous Trees

sive algorithms, and are better able to take advantage of the more obvious best contractor location possible in the sublinear case. The other algorithms were fairly stably in their relative utility.

For the case of homogeneous trees we found that the algorithms' results were much closer together, and that the low-exploration ϵ-first and PEEF algorithms perform much closer to the adaptive algorithms as budget increased. This indicates that more aggressive algorithms tended to perform better when the other contractors in their sub-trees finished exploration at the same time that they did.

Finally, we also considered the completion time of each algorithm as standard deviation of task completion time of the bottom level contractors in the tree varied, as seen in Figure 4. Here, we considered the linear utility distribution case only, and found that, intriguingly, the same pattern applied. This observation reinforced the conjecture that the relative performance of the algorithms was directly related to the extent of their exploration. Increasing the contractor's standard deviation, and effectively bringing the agents' time distributions closer together, had the same effect as increasing the amount of interaction required to discriminate effectively between the contractors – the algorithms that acted more aggressively, `PEEF` and ϵ-first ($\epsilon = 0.1$), became worse over time than the more deliberate algorithms KDE and ϵ-first ($\epsilon = 0.25$), while the adaptive `SOAAv` and `lS` algorithms performed the best overall.

6.2 Terraced Distributions for Base-variation Trees

We also experimented with terraced arm configurations (see Figure 5), which provided key insights into the relative merits of these algorithms. We constructed a case where we used 1 best contractor ($\mu_i = 10$), 4 good contractors ($\mu_i = 15$), and 5 poor contractors ($\mu_i = 25$), with $\sigma_i = 20$ for all but the best and $\sigma_i = 0$ for the best. We observed that the high-exploration ϵ-first algorithm and KDE was able to perform significantly better than any of the others at high budgets. The others performed somewhat worse, but retained the same ranking as before. What this case corroborated was the hypothesis that in cases where the best contractor has close competitors, more extensive exploration periods are more beneficial in the long-term (i.e., high budget) trials, even in the presence of the risk of contracting poor contractors that penalize continued sampling.

7. CONCLUSION AND DISCUSSIONS

In this paper we have evaluated the use of existing and recently-developed algorithms for solving the Fixed-Cost Multi-Armed Bandit Problem with Budget Constraint (MAB-BF) for addressing the core comprehensive trust management issues of engagement and evaluation of long-term reliability, and hence trustworthiness, of potential partners. We used a supply-chain domain where contractor agents have to depend on sub-contractors of initially unknown reliability to be able to fulfill their own production targets with the goal of minimizing time-to-completion of their orders. Algorithms that engage in strategic engagement, being cognizant of both current reliability estimates and remaining budgets for engagement before making final partner selections, have been used to develop trust estimates in potential long-term business partners.

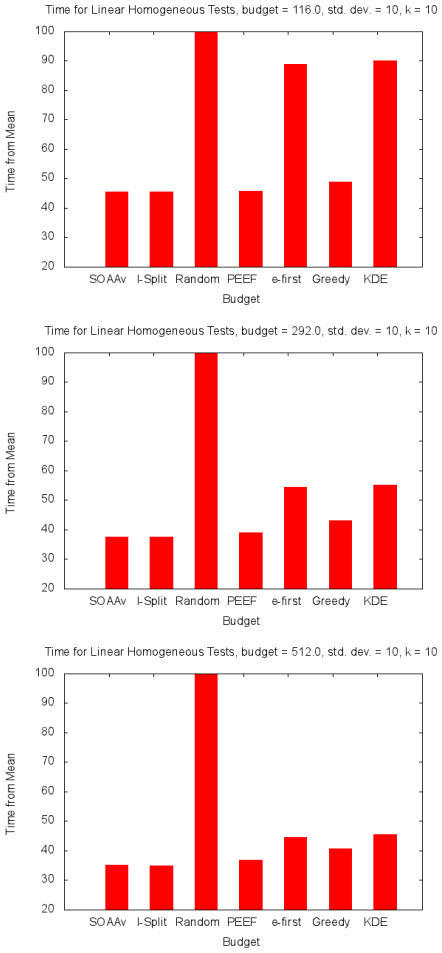

Figure 3: Time from Mean for Linear, Homogeneous Trees - at Low (left), Moderate (center), and High Budgets (right)

Our experimental evaluation of the proposed algorithms show that our proposed algorithms are successful in discriminating between partners of variable reliability in the face of domain uncertainty, in the form of stochastic performance variation over multiple levels of the supply-chain, by effectively managing exploration budget and partial reliability estimates. Experiments over a wide range of problem scenarios show that the `lS` and `PEEF` methods' improvement on the weaknesses of the ϵ-first algorithm allowed them to be highly competitive in tightly-constrained cases where other algorithms with better theoretical bounds cannot excel. At the same time, the `SOAAv` algorithm was able to perform very well in general, consistently outperforming Greedy along with the others in the majority of our test cases. Surprisingly, the simple Greedy

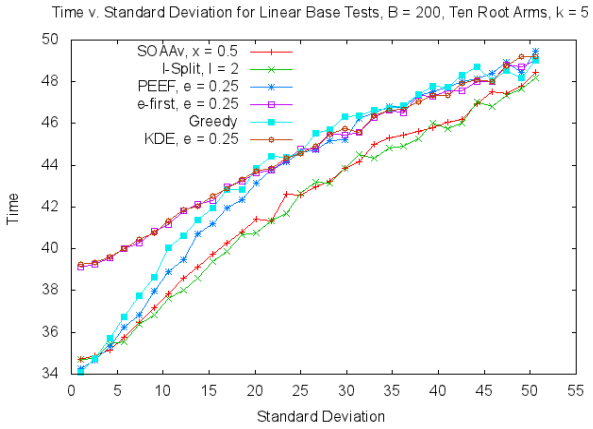

Figure 4: Time v. Standard Deviation for the Linear Case

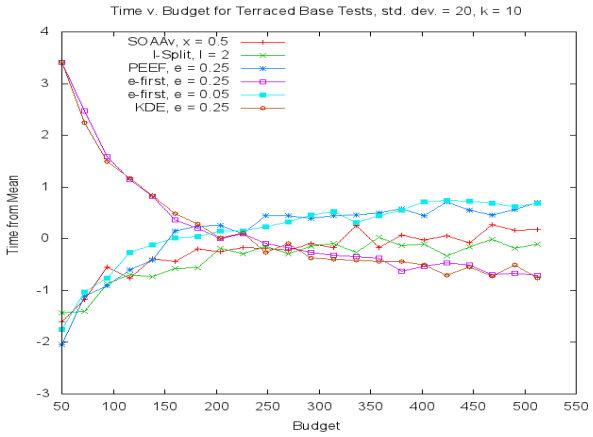

Figure 5: Time v. Budget for the Terraced Case

algorithm, not used in the literature earlier for the MAB-BF problem, held its own, underscoring the usefulness of maximizing the exploitation phase in most scenarios.

We plan to expand on this initial promising results of using MAB-BF algorithms for engagement and trust evaluation in a supply-chain domain by studying cases for variable, rather than uniform, topological configurations for each contractor below the first level in the supply-chains. It could be fruitful to study the effects of using algorithms in entire sub-trees, or entire levels of the tree.

We would like to develop variants of the proposed algorithms for the variable cost MAB problems which would be useful when contracting costs vary between contractors. We also plan to study an integrated approach that address, together with engagement and evaluation, other aspects of comprehensive trust management, e.g., the strategic use of known trustworthiness of contractors future contractor selection given expectations of upcoming task load distributions. Finally, we would like to characterize the applicability of the MAB-BF model and algorithms for trust management in diverse real-life problems.

Acknowledgments

Anton Ridgway and Michael Ripley were funded in part by the Tulsa Undergraduate Research Challenge (TURC) program at The University of Tulsa.

REFERENCES

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finitetime analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[2] R. Agrawal, M. Hedge, and D. Teneketzis. Asymptotically efficient adaptive allocation rules for the multiarmed bandit problem with switching cost. *Automatic Control, IEEE Transactions on*, 33(10):899–906, 1988.

[3] D. Ardagna, B. Panicucci, and M. Passacantando. A game theoretic formulation of the service provisioning problem in cloud systems. In *Proceedings of the 20th international conference on World wide web*, pages 177–186. ACM, 2011.

[4] C. Borgs, J. Chayes, N. Immorlica, K. Jain, O. Etesami, and M. Mahdian. Dynamics of bid optimization in online advertisement auctions. In *Proceedings of the 16th international conference on World Wide Web*, pages 531–540. ACM, 2007.

[5] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. In *Cloud Computing Technology and Science (Cloud-Com), 2011 IEEE Third International Conference on*, pages 304–311. IEEE, 2011.

[6] T. Chakraborty, E. Even-Dar, S. Guha, Y. Mansour, and S. Muthukrishnan. Selective call out and real time bidding. In *Internet and Network Economics*, pages 145–157, 2010.

[7] C. Castelfranchi and R. Falcone. Principles of trust for MAS: Cognitive autonomy, social importance, and quantification. In *ICMAS-98* pages 72–79, Los Alamitos, CA, 1998. IEEE Computer Society.

[8] W. Ding, T. Qin, X.Zhang, and T. Liu. Multi-armed bandit with budget constraint and variable costs. In *AAAI-13*. AAAI Press, 2013.

[9] D. Gambetta. *Trust*. Basil Blackwell, Oxford, 1990.

[10] S. Guha and K. Munagala. Approximation algorithms for budgeted learning problems. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 104–113. ACM, 2007.

[11] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006.

[12] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[13] S. Sen. A comprehensive approach to trust management. In *AAMAS'13*, pages 797–800, 2013.

[14] L. Tran-Thanh, A. Chapman, J. Munoz De Cote Flores Luna, A. Rogers, and N. Jennings. Epsilon-first policies for budget-limited multi-armed bandits. In *AAAI-10*, pages 1211–1216, 2010.

[15] L. Tran-Thanh, A. Chapman, A. Rogers, and N. Jennings. Knapsack based optimal policies for budget-limited multi-armed bandits. In *AAAI-12*, pages 1134–1140, 2012.

[16] P. Yolum and M. P. Singh. Engineering self-organizing referral networks for trustworthy service selection. *IEEE Transactions on System, Man, and Cybernetics*, 35(3):396–407, 2005.

[17] J. Zhang, R. Cohen, and K. Larson. Combining Trust Modeling and Mechanism Design for promoting Honesty in E-Marketplaces. *Computational Intelligence* 28(4): 549-578 (2012).