A Globally Optimal Algorithm for TTD-MDPs

Sooraj Bhat[1], David L. Roberts[1], Mark J. Nelson[1], Charles L. Isbell[1], Michael Mateas[2]
[1] College of Computing, Georgia Institute of Technology
[2] Department of Computer Science, University of California—Santa Cruz
{sooraj, robertsd, mnelson, isbell}@cc.gatech.edu, michaelm@cs.ucsc.edu

ABSTRACT

In this paper, we discuss the use of *Targeted Trajectory Distribution Markov Decision Processes* (TTD-MDPs)—a variant of MDPs in which the goal is to realize a specified distribution of trajectories through a state space—as a general agent-coordination framework.

We present several advances to previous work on TTD-MDPs. We improve on the existing algorithm for solving TTD-MDPs by deriving a greedy algorithm that finds a policy that provably minimizes the global KL-divergence from the target distribution. We test the new algorithm by applying TTD-MDPs to *drama management*, where a system must coordinate the behavior of many agents to ensure that a game follows a coherent storyline, is in keeping with the author's desires, and offers a high degree of replayability.

Although we show that suboptimal greedy strategies will fail in some cases, we validate previous work that suggests that they can work well in practice. We also show that our new algorithm provides guaranteed accuracy even in those cases, with little additional computational cost. Further, we illustrate how this new approach can be applied online, eliminating the memory-intensive offline sampling necessary in the previous approach.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Markov processes*; J.5 [**Computer Applications**]: Arts and Humanities—*Fine arts*

General Terms

Algorithms, Theory, Performance

Keywords

Markov decision processes, interactive entertainment, convex optimization

1. INTRODUCTION

Multi-agent systems often require coordination to ensure that a multitude of agents will work together in a globally coherent manner. There are a number of strategies for such coordination, ranging from a self-organizing decentralized system to a completely centralized system where one agent directly controls a set of automatons. We consider the case where agents are mostly autonomous, but are occasionally given instructions from a central coordinator. This architecture is a natural fit for a number of domains, including our motivating domain, drama management in interactive games.

Interactive games are frequently populated by non-player characters (NPCs) that interact with players and are often central to the storyline. While NPCs ought to be believably autonomous and appropriately interactive, it is extremely difficult for an author to create a successful game purely by authoring individual NPCs. The possible interactions among the NPCs themselves, to say nothing of the player, are difficult to predict and control in games of any reasonable size. A *drama manager* addresses this problem by acting as a central point of control, watching the progress of the game and directing the agents' activity to ensure a coherent story. While some drama-management systems control the game's progress fairly directly [10, 23, 11], a number take the approach of having mostly autonomous characters that are sometimes given direction by a central coordinator [1, 22, 5, 12, 13, 8, 17].

One line of work [1, 22, 5, 12, 13]—known as declarative optimization-based drama management (DODM)—poses drama management as an optimization problem. The drama manager has a set of actions it can take to intervene in the world, and a way of evaluating the quality of stories. The optimal actions are the ones that maximize expected story quality. Although this approach displays promising results, one problem is that if the drama manager is *too* effective in its task, it may manage to bring about the same optimal story each time, severely limiting replayability. One attempt to address the problem [12] boosts the rating of stories in which the player has many opportunities to change the direction of the story. This approach increases the chances that the player will see a different story in repeated plays, at least if she plays differently. On the other hand, a player who repeats many of the same actions might notice the same game response, giving the undesirable impression of a highly predictable game.

A more direct approach is to start with a distribution over possible stories and optimize the drama manager's use of actions so that it comes as close as possible to the target distribution. A framework for addressing this problem is *Targeted Trajectory Distribution Markov Decision Processes* (TTD-MDPs). Given an underlying Markov Decision Process (here, an abstract model of the story space) and a desired distribution over trajectories through its state space (here, a distribution of stories), a solution to a TTD-MDP is a stochastic policy that chooses the actions that result in a distribution of trajectories as close as possible to the target [18].

The original algorithm for solving TTD-MDPs performs quite well empirically, but suffers from two notable drawbacks. First, the solution to a TTD-MDP in any moderately complex domain requires sampling a tree of possible trajectories. Computing the full tree may not be feasible. Sampling only a part of the trajectory space can lead to "falling off" the trajectory space where a policy has been computed, necessitating some other online solution for recovery. Second, current solutions seek only to minimize local policy error at each place where the agent coordinator can take an action, rather than finding actions that are globally optimal. Worse, the current solutions do not even provably minimize local error.

Our main contribution in this paper is to show that one can minimize the global KL-divergence between the target distribution and the distribution induced by a TTD-MDP policy, using a procedure that only optimizes locally. We show that the empirical performance gain is only minor in our test domains, but the optimal solution can be guaranteed with little additional computational effort. Using this new procedure, we can also identify conditions under which one can efficiently compute the global optimum in an *online* fashion, eliminating the need for memory-intensive sampling. Thus, we hope that we can apply TTD-MDPs to a larger class of agent-coordination problems.

In the next sections, we provide a detailed overview of previous work on both drama management and TTD-MDPs. We then derive the KL-optimal solution to a TTD-MDP and discuss its online variant. We present an empirical comparison of the KL-optimal approach with two variants of previous algorithms. Finally, we conclude by discussing related and future work.

2. BACKGROUND

Here, we describe *declarative optimization-based drama management* (DODM) as well as TTD-MDPs. In describing DODM, we aim to provide context for the use of TTD-MDPs. In describing TTD-MDPs, we wish to introduce the technical details necessary to understand our contribution.

2.1 Drama Management

In DODM, a story is represented as a sequence of discrete *plot points* [13]. In this framework, there is a *drama manager* (DM) that has a set of requests it can make of non-player characters (NPC) that are concretely implemented as agents in a game world. The DM acts as coordinator for the NPC agents.

Plot points represent important events that can occur in the story. They may involve a player uncovering information, finding an object, or moving to a location. Plot points are abstractions of concrete events happening in the game world. Finding a safe could involve a number of concrete actions by a player, such as entering the room in which it's located, exploring the room, possibly examining various objects in the room, searching the bookshelf on the wall, and finally moving the book behind which the safe is hidden. The drama manager does not need to know the details of all these actions, as most do not impact the progression of the story; instead, it only senses the abstract plot point. Plot points also have precedence constraints. For example, the end must occur after the beginning, and a safe cannot be opened without its key. Note that not all plot points need occur during any particular game episode.

The set of DM actions is exactly the the set of requests to which an NPC agent can respond to, as implemented by the game author. For example, the DM can request an NPC to begin or avoid a conversation, request a door to be unlocked or locked, and so on. DM actions are also abstract and constrained. For example, the DM action *hint-safe-location* should not be applied after the safe has been found. Further, *hint-safe-location* may be realized as an entire series of concrete actions by an NPC agent (*e.g.*, drawing a player into a conversation in which a hint may be dropped). Of importance to the drama manager is the *expected outcome* of taking an action. Typically, DM actions can take one of three forms: *cause*, *deny*, or *hint*. A cause action causes its target to happen with probability 1.0. When a particular plot point is denied by the DM, it is assumed that it will never happen. When the DM employs a hint action, the result is a modification of the distribution over player-caused plot-points toward the plot point that is hinted by the DM.

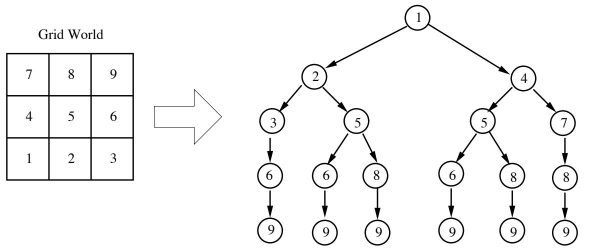


Figure 1: A 3×3 gridworld with deterministic actions *Right* and *Up*, along with the resulting trajectory tree.

DODM also requires an author-supplied evaluation function. This function takes as input the sequence of plot points and history of DM actions and outputs a quality score. Through this evaluation function, the author specifies what it means to be a good story. In practice, the evaluation function is a linear combination of features over plot point sequences. For example, the author of a mystery game may assign a higher score to sequences where tension is built slowly followed by a rapid burst of revelations.

The specific details of how plot points, DM actions, and evaluation functions are specified in a concrete game are beyond the scope of this paper; however, it is important to understand that all of these components relate directly to the specification of an MDP: **States** correspond to sequences of plot points in DODM; **Actions** correspond to DM actions; **Transitions** correspond to a probabilistic user model; and **Rewards** correspond to story evaluations.

2.2 TTD-MDPs

A traditional MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $P : \{\mathcal{S} \times \mathcal{A} \times \mathcal{S}\} \to [0, 1]$ is a transition function, and $R : \mathcal{S} \to \mathbf{R}$ is a reward function. The solution to an MDP is a policy $\pi : \mathcal{S} \to \mathcal{A}$. An optimal policy ensures that the agent receives maximal long-term expected reward.

A TTD-MDP is similarly defined by a tuple $(\mathcal{T}, \mathcal{A}, P, P(\mathcal{T}))$, with states $\mathcal{T}$ that are (possibly partial) finite-length trajectories of MDP states; a set of actions $\mathcal{A}$; a transition model P; and a target distribution over complete trajectories $P(\mathcal{T})$. The target distribution in a TTD-MDP replaces the reward function in a traditional MDP. The solution to a TTD-MDP is a policy $\pi : \mathcal{T} \to P(\mathcal{A})$ providing a distribution over actions in every state. The optimal policy results in long-term behavior as close to the target distribution as possible (we define "closeness" more rigorously later).

Any finite-length discrete-time MDP can be converted to a TTD-MDP.[1] Consider an MDP with a set of states $\mathcal{S}$ and sets of actions available in each state $\mathcal{A}_s$, the probability $P_{i+1}(s')$ that the process is in state s' at time $i+1$ is defined recursively by:

$$P_{i+1}(s') = \sum_{\forall s \in \mathcal{S}, a \in \mathcal{A}_s} \left(P(s'|a, s) \cdot P(a|s) \cdot P_i(s)\right) \qquad (1)$$

where $P(s'|a, s)$ is the transition model encoding the dynamics

[1]Actually, we can solve infinite length TTD-MDPs under certain conditions; however, this capability has not been necessary for any of the applications of TTD-MDPs that we have considered so far.

of the world and $P(a|s)$ is the policy under the agent's control (usually written as $T(s', a, s)$ and $\pi(s', a)$, respectively). If we assume (as is commonly done) that the policy is deterministic, we obtain a common form of Equation 1, rewritten as: $P_{i+1}(s') = \sum_{\forall s \in \mathcal{S}} P(s'|\pi(s), s)$.

Because in TTD-MDPs we are interested in trajectories, we can simply consider the history of the MDP states as the TTD-MDP trajectories, resulting in a TTD-MDP where each trajectory represents a sequence of states in the underlying MDP, optionally including a history of the actions taken. Dealing with trajectories also means that the "state" space of the TTD-MDP forms a tree. We can thus restate Equation 1 for TTD-MDPs:

$$P(t') = \sum_{\forall a \in \mathcal{A}_t} \left(P(t'|a, t) \cdot P(a|t)\right) \cdot P(t). \tag{2}$$

In other words, for every trajectory t', $P(t'|a, t)$ is nonzero for exactly one $t \prec t'$ that is its prefix. This observation follows from the fact that each trajectory represents a unique sequence of states $s_1, \ldots, s_{\|t\|}$ and therefore has a unique prefix. Thus, the summation need only account for possible actions taken in the preceding trajectory rather than actions in multiple MDP states. Because each trajectory has a fixed length and can therefore appear at only one specific time, we can drop the i subscripts.

Trajectories represent total history traces of an online decision making process. Consider Figure 1, a 3×3 gridworld where there are two deterministic actions: *move right (R)* and *move up (U)*. The right side of the figure depicts the corresponding *trajectory tree*. A trajectory tree is simply a graphical representation of the valid trajectories and the prefix relationship that governs partial trajectories.

In this example valid trajectories have initial state 1 and terminating state 9. Let us first consider the partial trajectory $1 \xrightarrow{R} 2$. It is a prefix of two immediate subsequent partial trajectories ($1 \xrightarrow{R} 2 \xrightarrow{R} 3$ and $1 \xrightarrow{R} 2 \xrightarrow{U} 5$) as well as three other partial trajectories and three complete trajectories.

Note that a state in the underlying world may appear multiple times in the trajectory tree yet participate in distinct trajectories. Consider both appearances of state 5, for instance. The one on the left represents the partial trajectory $1 \xrightarrow{R} 2 \xrightarrow{U} 5$ whereas the one on the right represents the partial trajectory $1 \xrightarrow{U} 4 \xrightarrow{R} 5$; they depict the same world *state*, but they are different *trajectories* because the process arrived to that state along a different path. The power of considering trajectories rather than states is we can make decisions based on not only where you are but also how you got there.

To complete the TTD-MDP specification, we need a target distribution. There are a variety of ways one might imagine deriving such a distribution; however, in this work we will focus on the case where a distribution has been pre-specified and may be queried (although we do not assume we can sample from it).

3. MINIMIZING ERRORS IN TTD-MDPS

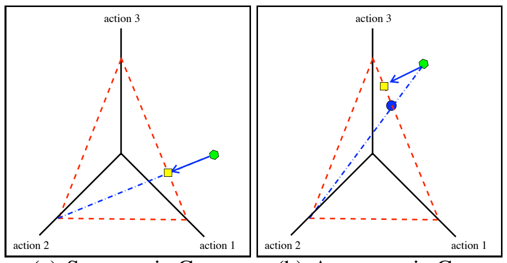
Unfortunately, it is not always possible to find a stochastic policy that exactly solves a TTD-MDP. Imagine that for a given trajectory, t', we have two trajectories we may reach, t_1 and t_2, and two actions we may use, a_1 and a_2. Our target is $P(t_1) = 1.0$ and $P(t_2) = 0.0$. If both a_1 and a_2 have non-zero probability of transitioning to t_2 from t' then it is impossible to construct a distribution over a_1, a_2 that will yield our desired distribution over t_1, t_2. In this section we will review a previous attempt to address this problem, and derive an algorithm that provably minimizes global error.

3.1 A Previous Attempt

The original formulation of TTD-MDPs used Algorithm 1 to

Algorithm 1 Algorithm to minimize local L_1.

```
1: Build a (complete if possible) tree of trajectories. If a complete tree is
   not possible, sample the tree from the target distribution.
2: Initialize each leaf node (complete trajectory) with its target probability
   $P(t)$.
   In reverse topological order:
3: for Every $t$ do
4:     for Every child $t'_i$ of trajectory $t$ do
5:         Condition Equation 2 on $t$:
               $P(t'_i|t) = \sum_{\forall a \in \mathcal{A}_t} (P(t'_i|a, t) \cdot P(a|t))$
6:     end for
7:     This forms a system of $|\mathcal{T}_{t'_i}|$ linear equations in $|\mathcal{A}_t|$ unknowns:
               $\vec{P}(t'_i|t) = \vec{P}(t'_i|a, t) \cdot \vec{\pi}_t$
       which can be solved for $\pi$ using standard linear algebra.
8: end for
```



(a) Symmetric Case (b) Asymmetric Case

Figure 2: Geometric view of the normalization procedure.

construct a policy $\pi_t(a) = P(a|t)$ for every partial trajectory in $\mathcal{T}$. When an exact solution exists, the algorithm will return an optimal policy; however, when this is not possible, undesirable behavior can result. For example, consider a partial trajectory t', three subsequent trajectories t_1, t_2, t_3 and three actions a_1, a_2, a_3 whose linear system is defined by:

$$\begin{bmatrix} 0.0 \\ 0.3333 \\ 0.6667 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0.0 \\ 0.0 & 0.5 & 0.5 \\ 0.5 & 0.0 & 0.5 \end{bmatrix} \cdot \vec{\pi}_t \tag{3}$$

The solution vector

$$\vec{\pi}_t = \begin{bmatrix} 0.3333 \\ -0.3333 \\ 1.0000 \end{bmatrix}$$

does not represent a probability distribution.

While this solution is not a vector of probabilities, it does have an interpretation: achieving the desired distribution requires that a_2 be "undone" some percentage of the time. Because that is impossible, the original algorithm replaces any negative values with zeros, renormalizing the resulting vector to create a distribution.

The procedure is an attempt to quickly minimize the L_1 error measured over $\|\widehat{Y} - Y\|_1 = \|\vec{P}(t_i|a, t) \cdot \widehat{\pi}_t - Y\|_1$ where $\widehat{Y}$ is the distribution of trajectories for a particular choice of policy $\widehat{\pi}_t$ and Y is the desired distribution. Although the procedure performs well empirically, it is not guaranteed to minimize L_1 error.

Figures 2(a) & 2(b) illustrate why. These figures provide a geometric interpretation of the normalization step. The solution space to the linear system lies somewhere on a hyperplane constrained by the transition matrix. Because we want the solution to be probabilities, the region of valid solutions on this hyperplane must lie somewhere between 0 and 1 on each of the axes. In the case presented in Equation 3, we have a situation where the probability restriction is not met. This is depicted in the figures by the green hexagon farthest to the right. In Figure 2(a) we depict a situation where symmetry in the transition matrix results in an optimal solution and in Figure 2(b) we depict a situation where this symmetry does not exist. The normalization procedure is represented by the dashed blue line from the solution dot to the action which received negative mass. The intuition here is that by assigning zero mass to the action whose solution is negative and normalizing we are maintaining the probabilities of the other two actions in the same relative proportion. On the other hand, the optimal solution is the point in the valid region of the hyperplane that is closest to the actual solution (*i.e.* lies on a perpendicular line from the boundary of the hyperplane to the solution point). This optimal result is depicted by the yellow square that lies on the hyperplane boundary. The optimal answer is depicted by the blue arrow. Note that in the symmetric case, the location of the optimal solution and the normalized solution are the same. In the asymmetric case, the location of the normalized solution does not coincide with the optimal solution.

To achieve true local optimality, it is necessary to solve a constrained optimization of the L_1 error. The objective function is:

$$\min_{\vec{\pi}_t} \|\vec{P}(t'_i|t) - \vec{P}(t'_i|a,t) \cdot \vec{\pi}_t\|_1 \tag{4}$$

subject to $\sum_a \pi_t(a) = 1$ and $0 \leq \pi_t(a) \leq 1$. Standard techniques exist to reduce this objective function to a constrained linear program [2]. This optimization procedure replaces Step 7 in Algorithm 1 and is guaranteed to minimize *local* L_1 error.

3.2 A KL-optimal Algorithm

Although we now have a procedure to minimize local L_1 error, no guarantees can be made regarding global error. In this section, we present an algorithm for solving TTD-MDPs based on minimizing the *Kullback Liebler divergence*. The KL-divergence between two probability distributions $p(x)$ and $q(x)$ is defined as:

$$\begin{aligned} D_{KL}(p\|q) &= \sum_x p(x) \log \frac{p(x)}{q(x)} &(5)\\ &= \sum_x p(x)\log p(x) - \sum_x p(x) \log q(x) &(6)\end{aligned}$$

KL-divergence is not a true distance, as it is asymmetric; however, it is a well-understood measure with several important properties. In particular, it is consistent, always non-negative and zero only when p and q are equal. If we think of p as a base distribution then D_{KL} measures how well q approximates it, by measuring the entropy that remains in q. In our case, p is the target distribution and q is our approximation of it. In TTD-MDPs q is:

$$\begin{aligned} q(\tau) &= \prod_{t \preceq \tau} w(t) &(7)\\ \text{where} \quad w(t') &= \sum_a P(t'|a,t) \cdot \pi_t(a) &(8)\end{aligned}$$

Here, τ represents complete trajectories while t and t' are (partial or complete) trajectories. The probability $w(t')$ represents the frequency with which t' is targeted when the process is at t. It combines information about the probabilistic policy and world dynamics at t. Thus, the product of these one-step transition frequencies, $w(t')$, yields the probability of a complete trajectory, $q(\tau)$. Our objective function for optimization then becomes:

$$\begin{aligned} \max_\pi \sum_\tau p(\tau)\log q(\tau) &= \max_\pi \sum_\tau p(\tau) \log \prod_{t\preceq\tau} w(t)\\ &= \max_\pi \sum_\tau p(\tau) \sum_{t\preceq\tau} \log w(t)\\ &= \max_\pi \sum_\tau \sum_{t\preceq\tau} p(\tau)\log w(t)\end{aligned}$$

Note that a partial trajectory t contributes $p(\tau)\log w(t)$ to the sum for each complete trajectory τ for which it is a prefix. We can define a function over complete trajectories summarizing the factor of $\log w(t)$ that t contributes, $m(t) = \sum_{t\preceq\tau} p(\tau)$. Our objective function is then:

$$\max_\pi \sum_t m(t) \log w(t) \tag{9}$$

Note that $m(t)$ represents the total probability mass contained in the subtree rooted at t. Now, obtaining the optimal policy is simply a matter of performing an argmax_π on the objective function.

Having summarized and isolated the contribution of an individual trajectory t' to the objective, we could proceed using a naive approach and optimize for each trajectory independently, thus optimizing their sum; however, this procedure ignores the fact that optimizing for one trajectory may come at the cost of sacrificing the optimality of a sibling trajectory. Fortunately, optimizing for a trajectory t' is only constrained by the optimization of its sibling trajectories, and no others because the local stochastic policy must satisfy a sum-to-one constraint. This insight enables us to consider trajectories by groups of siblings—precisely the same grouping used by the earlier approach that sought to minimize local L_1 error. Fortunately, we have two distinct advantages: 1) the groups of siblings can be solved for in *any* order (there is no restriction that you must start with the leaves and work towards the root); and 2) solving the local optimizations is guaranteed to produce the globally optimal setting of the parameters. The local optimization is:

$$\begin{aligned} &\operatorname*{argmax}_{\pi_t} \sum_{t\to t'} m(t')\log w(t') &(10)\\ =\ &\operatorname*{argmax}_{\pi_t} \sum_{t\to t'} m(t') \log \sum_a P(t'|a,t)\cdot\pi_t(a) &(11)\end{aligned}$$

where we have used $t \to t'$ to indicate that t' is a child of t. This objective is convex, so Equation 10 can be solved using a standard technique for constrained convex optimization [2]. Thus, we yield a KL-optimal offline algorithm by replacing Step 7 of Algorithm 1 with this local KL-based optimization.

One potential problem arises when $q(\tau)$ is forced to be zero for a complete trajectory τ with $p(\tau) \neq 0$. This occurs when no actions available at a trajectory t will move us to a child t', *i.e.* $w(t') = 0$. $D_{KL}(p\|q)$ is undefined due to the division by zero. Thus all possible approximations seem equally bad, preventing us from making progress towards a solution. Luckily, this problem can be eliminated by preprocessing and reformulating each local optimization to eliminate child trajectories t' that can never be reached. Intuitively, t' should not be represented in the trajectory tree if it cannot ever be reached from t. In fact, in most domains (including all of the domains we have used) this issue does not arise at all because of the process by which the trajectory trees are generated.

3.3 An Online Algorithm

We now describe how this approach can be applied in an online fashion, eliminating the need for the potentially costly offline sampling step employed by earlier techniques. For example, in the drama management domain, it would be undesirable to have to construct extremely large trajectory trees on the player's machine.

As formulated, we require $m(t)$ to be available for each local optimization. The requirement is actually weaker—all that is necessary is a function that, for a given parent trajectory t, gives the *relative* masses required for each child of t.

Below we derive an online algorithm when $m(t)$ can be computed quickly; however, we note that even when $m(t)$ cannot be computed efficiently, the resource requirements of the KL-based offline algorithm are no worse than in previous work. By processing the trajectories in the same order as before (*i.e.* propagating upwards from the leaves) we are still able to calculate the $m(t)$ values as we need them, because of the following properties:

$$m(t) = \sum_{t \to t'} m(t') \tag{12}$$

$$m(\tau) = p(\tau) \tag{13}$$

which follow from the definition of $m(t)$. In the event that the whole trajectory tree does not fit into memory, we can employ the same sampling technique used in the earlier approach.

To derive the online algorithm, one only needs to recall that we achieve the globally KL-optimal solution regardless of the order in which we perform the local optimizations. One could start by processing the root of the trajectory tree, then process the children of the root, and so on, to compute a policy for each node in the tree. If $m(t)$ can be computed efficiently (or in the background), then we can do even better by only solving the local optimizations that we encounter during an episode. This is done by interleaving the local optimization steps with taking actions in the world—the local optimization tells us how to take the next action, and the action places us at a new node in the trajectory tree. Thus, we only solve the local optimization for trajectories we actually encounter along a particular trajectory (root-to-leaf path in the trajectory tree), rather than *all* optimizations in the trajectory tree. In terms of number of optimizations to solve, we never do worse than the previous approach (as the size of any trajectory is bounded by the size of the tree), and typically we will do much better—for a fairly balanced and complete tree with height h and branching factor b, a trajectory will have size h whereas the tree will have size roughly equal to b^h.

Algorithm 2 Online algorithm to minimize global KL error.

```
1: $t \leftarrow$ start state
2: while $t$ is not a complete trajectory do
3:     Compute the optimal local stochastic policy $\pi_t^*$:
           $\pi_t^* = \operatorname{argmax}_{\pi_t} \sum_{t \to t'} m(t') \log \sum_a P(t'|a,t) \cdot \pi_t(a)$
4:     Sample an action $a$ from $\pi_t^*$.
5:     Take action $a$ in the world, which will transition probabilistically to
       a new trajectory $t'$ according to $P(t'|a,t)$.
6:     $t \leftarrow t'$
7: end while
```

Note there is no free lunch: without extra information about the nature of the trajectories or the analytic structure of $p(\tau)$, we are still limited by the complexity of the summation to compute $m(t)$ from the tree. Below we briefly discuss a few approaches to make the online algorithm feasible.

First, we could require $m(t)$ to be of a certain easily computable form. Specifically, we could require that $p(\tau)$ be constructed in such a way so that $m(t)$ can be computed solely from a partial trajectory t. Consider a simple game exactly 10 plot points in length and with a fixed branching factor. Suppose we would like a drama manager to target only stories that contain a certain plot point A with uniform probability. Given a partial trajectory t, we can tell whether or not A has occurred through simple inspection. If A *has* occurred, we know that all complete trajectories subsequent to t will have non-zero mass. Thus, via a counting argument, we can efficiently determine $m(t)$. If A *has not* occurred, because all stories occur with uniform probability, we can employ a combinatorial argument to calculate the number of subsequent trajectories that contain A. The applicability of this authorial idiom to drama management is the subject of ongoing research.

Another approach is to use *online sampling*. An estimate $\hat{m}(t)$ of $m(t)$ can be computed by constructing an approximate trajectory tree built from simulated complete trajectories collected in a background thread. This approach leverages the fact that $\hat{m}(t)$ only needs to provide *relative* values. As such, the samples need only provide a good representation of the space, rather than provide a perfect estimate of $m(t)$. Initial experiments on this approach are promising.

4. RESULTS

In this section, we present results that examine the solution quality and performance characteristics of the proposed algorithm versus the previous algorithm as well as a baseline algorithm.

4.1 Solution Quality

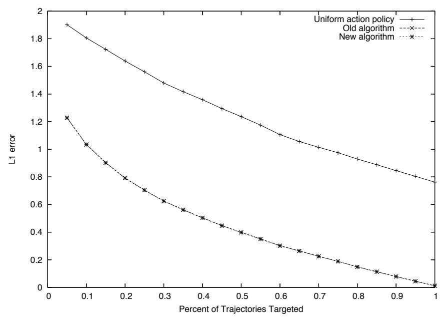
To verify that the KL approach performs well in practice, we ran experiments on a synthetic grid world and on the drama management MDP studied in [13]. The synthetic grid world is the same discussed in Section 2.2 and shown in Figure 1. As a reminder, there are at most two actions in every state ("move right" and "move up") and a known transition model $P(s'|a,s)$.

For each trial we selected a given complete trajectory τ with probability δ. Each selected trajectory was given uniform target probability and the remainder given zero target probability. We varied δ from 0.05 to 1.0. We then built the tree and computed the policy according to the local L_1 approach, the local KL approach, and a simple algorithm that implements a uniform policy. We varied the size of the grid from 5×5 to 9×9. For each set of parameters, we ran 10 trials and averaged the results.

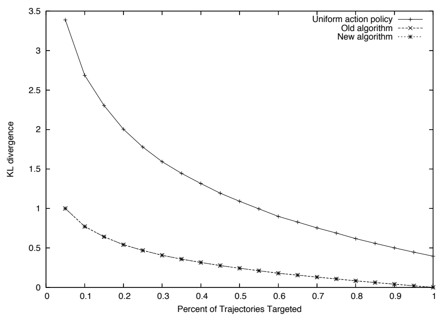
The results are presented in Figure 3. As the percentage of trajectories targeted increases, both L_1 and KL error decrease. This is expected, as any nondeterminism in action outcomes can result in a terminal trajectory with zero desired probability, so reducing the number of zero mass terminal trajectories reduces error. Note that the local L_1 and KL approaches perform nearly identically. As expected, the baseline algorithm does far worse than the other two, thus showing that at least the domain is not trivial.

To test on a computationally intractable problem, we used a version of the *Anchorhead* story previously explored using reinforcement learning [13]. In our model, *Anchorhead* has 29 plot points and 90 drama manager actions. The evaluation function is a combination of features, such as the spatial locality of action, having plot points occur in an order that motivates later events, and so on; more details on the general framework and evaluation functions are given by Nelson & Mateas [12].

To evaluate the effect of a drama manager, we run simulated stories and plot a distribution of story qualities. The drama manager's goal is to increase the frequency of highly-rated stories and decrease the frequency of low-rated stories. In particular, we would prefer that very low-rated stories never happen, while achieving a good distribution over highly-rated stories. We use the evaluation function and these preferences to define a TTD-MDP: any trajectory of plot points that evaluates below a threshold has a target fre-



(a) Plot of L_1 error



(b) Plot of KL error

Figure 3: Error plots for the 9×9 grid world. The old and new methods perform nearly identically, thus their plots overlap. The maximum difference between the two w.r.t. L_1 error is 2.48×10^{-6}; w.r.t. KL error, 9.17×10^{-9}. The uniform action policy is much worse in all case, by both measures.

quency of 0, while those above the threshold should all appear, with the highly-rated ones more likely.

We build a sampled trajectory tree to both estimate $m(t)$ by summation and to solve for the TTD-MDP policy. Unfortunately, during evaluation or actual gameplay, it is often the case that nondeterminism in our player model causes us to encounter story trajectories not in the tree. In this event, the fallback policy for the drama manager is to take no action for the remainder of the game. We opt to ignore the recovery mechanism considered by Roberts *et al.* [18] in favor of a "pure" comparison of the optimization methods.

Again, the results for the old and new algorithm were nearly indistinguishable, consistent with what we observed with the grid world. It is possible, however, to construct cases where *ad hoc* local optimization fails. The following examples use 3 actions and 3 successor states. We will use **l1-sub** to refer to the previous, suboptimal matrix-based solution; **l1-opt** to refer to the correct L_1-based solution outlined in Equation 4; and **kl-opt** to refer to the KL-based algorithm presented in this paper. We present the results of the three approaches on two pathological examples in Figure 4.

In the first example (Figure 4(a)), **kl-opt** does better than **l1-sub** w.r.t. L_1 error. In fact, the difference in L_1 error between the two is 0.5, which is 25% of the maximum possible L_1 error (2.0). In the second example (Figure 4(b)), though **kl-opt** cannot beat **l1-opt** w.r.t. L_1 error (by definition), it *does* do better than both **l1-opt** and **l1-sub** w.r.t. KL error and does equally as well on L_1 error.

In both examples, the local transition matrix makes it very difficult to reach the target distribution. The first row of the first transition matrix contains relatively large numbers compared to the target probability mass for the first row of the target probability vector. A similar situation occurs in the second example.

In the real-world domains we have examined, this situation does not seem to arise often. It could be that any local technique that attempts to match the local target distribution according to a reasonable error measure will perform quite well in practice on the sorts of problems we have explored; however, many such techniques will fail in exactly the difficult cases. By contrast, we have derived an algorithm that provides theoretical guarantees and, as we shall see, remains computationally feasible even within the performance constraints of an interactive drama management system.

4.2 Execution Time

Given that both the theoretically grounded technique and the ungrounded technique perform similarly in the types of domains we have studied, we seek to characterize the computational tradeoffs that are made when one algorithm is used over the other. The following experiments were run on an Intel Pentium Xeon Processor at 3.8GHz with a 2,048 KB cache and 8GB of RAM (although only 1GB was allocated to the Java 1.5.0 `server` JVM).

First, we examine the runtime of **l1-sub**, **kl-opt** and a uniform random policy on the grid world domain. Table 1 contains the results of those experiments. Note the rapid growth in running time for all of the approaches. This is due to the exponential increase in trajectory tree size as the size of the grid world increases. Specifically, for an $n \times n$ grid world, there are $\frac{(2n-2)!}{(n-1)!(n-1)!} = O(n!)$ local optimizations to be solved. In the case of **l1-sub**, each of the local computations involves solving a system of linear equations, which can be accomplished relatively efficiently—$O(a^3)$ for a naive implementation where a is the number of actions. On the other hand, the **kl-opt** technique requires a polynomial number of steps.[2]

	Total time (ms)			Normalized time (ms)		
Size	**uni**	**l1-sub**	**kl-opt**	**uni**	**l1-sub**	**kl-opt**
5×5	1.90	2.58	76.88	0.0271	0.0369	1.0983
6×6	7.22	8.10	233.16	0.0287	0.0321	0.9252
7×7	34.30	34.52	840.48	0.0371	0.0374	0.9096
8×8	161.02	167.50	3191.48	0.0469	0.0488	0.9299
9×9	769.06	789.70	12119.28	0.0598	0.0614	0.9417

Table 1: Average computation time for the uniform baseline, l1-opt, and kl-opt algorithms for various grid sizes.

As shown in Table 1, the run time for the **kl-opt** approach is significantly higher. To examine this effect more closely, consider the second column, which presents the normalized computation time (*i.e.* time per local optimization). Note how for the **kl-opt** approach, the normalized running time remains essentially constant for all grid sizes. However, for the baseline and the **l1-sub** approach, the normalized run time jumps significantly for the largest grid size (marked in bold in the table). We attribute this to cache misses. The increased time required per computation for the **kl-opt** approach hides the latency encountered for a cache miss.

Although **kl-opt** requires roughly thirty times the computation than **l1-sub** per decision point, we are well under what is necessitated by the domain; other researchers have allocated as much as two *seconds* between decision points for the drama manager to select an action. These results indicate that the local **kl-opt** approach can execute in less than a millisecond, easily meeting the

[2]The polynomial is a function of a number of parameters of the optimization and the condition number of the objective function's Hessian. A complete analysis is beyond the scope of this paper.

Equation:	$\begin{bmatrix} 0.0027 \\ 0.3586 \\ 0.6388 \end{bmatrix} = \begin{bmatrix} 0.7432 & 0.2535 & 0.5272 \\ 0.1626 & 0.2175 & 0.2172 \\ 0.0942 & 0.5290 & 0.2556 \end{bmatrix} \cdot \vec{\pi}_t$		
Method:	**l1-sub**	**l1-opt**	**kl-opt**
L_1 error:	1.0491	0.5017	0.5017
KL error:	0.7507	0.2875	0.2875
Solution vector:	$\begin{pmatrix} 0.0 \\ 0.0 \\ 1.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 1.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 1.0 \\ 0.0 \end{pmatrix}$

(a) Example 1

Equation:	$\begin{bmatrix} 0.4177 \\ 0.2182 \\ 0.3640 \end{bmatrix} = \begin{bmatrix} 0.1130 & 0.0025 & 0.0085 \\ 0.6178 & 0.5717 & 0.5559 \\ 0.2692 & 0.4258 & 0.4356 \end{bmatrix} \cdot \vec{\pi}_t$		
Method:	**l1-sub**	**l1-opt**	**kl-opt**
L_1 error:	0.8037	0.7286	0.7991
KL error:	1.1039	0.6444	0.4288
Solution vector:	$\begin{pmatrix} 0.0709 \\ 0.0 \\ 0.9291 \end{pmatrix}$	$\begin{pmatrix} 0.4304 \\ 0.0 \\ 0.5697 \end{pmatrix}$	$\begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \end{pmatrix}$

(b) Example 2

Figure 4: Two pathological examples where there is a quantifiable difference between the results of the three optimization approaches. The tables contain the equations that represent the local optimization, the solution vector found by each of the three methods, and the error associated with each of the solution vectors.

constraints of a real-time interactive drama management system. Further, this approach retains is optimal properties even when used online, so the increased time per local computation is only a linear penalty in practice.

5. RELATED WORK

The inspiration for this work is drawn from two fields: drama management and Markov decision processes.

Drama Management. Using a drama manager to guide interactive entertainment was first proposed in 1986 by Laurel [6]. The particular formalism based on plot points, DM actions, and an evaluation function that we use was proposed as *search-based* drama management (SBDM) by Bates [1]. There are other drama-management approaches: Mateas & Stern use a *beat-based* drama manager [10]; Mott & Lester use a decision-theoretic planner for their drama manager [11]; Young *et al.* use a goal-based planning drama manager [23]; and Magerko uses an architecture called IDS based on a "playwriting expert system" [8]. For a somewhat dated description of drama management techniques see Mateas' 1999 survey [9].

The SBDM formulation of the drama-management problem was posed as an optimization problem to be solved using a minimax game-tree-like search. It was studied extensively in the 1990s by Weyhrauch [22]. Weyhrauch discovered that full-depth search was intractable for any reasonable problem and proposed, SAS+, a technique that used static evaluation sampling.

Lamstein & Mateas proposed reviving the technique in 2004 [5] but later work by Nelson & Mateas showed that the approach did not scale to other story worlds. This result led to the development of DODM [13] where the sampling search was replaced with reinforcement-learning in the spirit of Tesauro's TD-Gammon [20, 19, 21]. Finally, Roberts *et al.* developed TTD-MDPs to address the issue of replayability through variety of experience [18].

TTD-MDPs. The formulation of TTD-MDPs as optimizing a probabilistic policy was inspired by a number of works from the MDP community. Specifically, Isbell *et al.* [3] developed *Cobot*, a social reinforcement-learning agent. In that work, Cobot was tasked with random action selection during episodes of user interaction. Cobot had "internal" models of user preferences represented by state-action Q-values that were used to induce a distribution over actions. This distribution results in action selection roughly proportional to user satisfaction but does not allow the targeting of specific distributions over actions or actions sequences.

Kearns, Mansour, & Ng [4] developed a method for approximate planning in POMDPs based on a sampled "trajectory tree". The concept of a sampled trajectory tree in that work differs from ours. Their trajectory trees are sampled from a generative model of observations and transitions and represent a single episode. Multiple trees are then combined to create estimated values for particular actions. In our case, the trajectory tree is sampled uniformly from the state of possible trajectories and represents possible paths through the space—not a single estimated path based on observations.

Littman [7] uses Markov games as a framework for exploring reinforcement learning in the presence of multiple adaptive agents. Our formulation of TTD-MDPs is similar in that it acknowledges a source of randomness outside the environment itself, finding a solution that requires a probabilistic policy. On the other hand, this work was not directly concerned with distributions over outcomes, and assumed a zero-sum game. In the formulation of the drama-management problem, the game is not zero-sum: the player and the drama manager are, in effect, playing completely different games.

Lastly, if we think of TTD-MDPs as modeling the dynamics of an MDP while holding non-traditional elements fixed, we find common ground with recent work in inverse reinforcement learning (IRL) [14] and extended Markov tracking (EMT) [15, 16]. In IRL, the goal is to observe the policy of an agent and infer a reward function that would lead to its behavior. In the drama management case, non-determinism arises from both the user and DM actions. If we think of the desired distribution as fixed, then the non-determinism that is not explained by the user model is exactly the distribution over our own actions. Similarly, in EMT, the goal is to select the optimal system dynamics (*i.e.* probability of state given another state) given a sequence of observations in a partially observable environment. Once the optimal dynamics have been identified, the task becomes the probabilistic selection of an action from a distribution that fits the desired dynamics.

6. CONCLUSIONS AND FUTURE WORK

Targeted Trajectory Distribution Markov Decision Processes are well-suited to be a framework for centrally coordinating multiagent systems, especially in cases where reusability of the system is important, such as interactive games. In this paper, we have identified some shortcomings of previous TTD-MDP algorithms, and derived a new algorithm that minimizes global KL-divergence. In addition, we discussed the conditions that enable efficient online variations of the algorithm.

As the probability distribution is of central importance to a good TTD-MDP-based system, it is necessary to make it easy for designers to specify target distributions. This can take a number of different directions. When converting from an optimization-based coordinator to a TTD-MDP-based one, a natural approach is to make the distribution be some function of the evaluation function as we have done; however, writing an evaluation function may not always be easy, and the distribution induced from one may not lead to the distribution the designer really intended. Thus, we are interested in pursuing methods for preference elicitation.

Acknowledgments

This research was performed while on appointment as a U.S. Department of Homeland Security (DHS) Fellow under the DHS Scholarship and Fellowship Program, a program administered by the Oak Ridge Institute for Science and Education (ORISE) for DHS through an interagency agreement with the U.S Department of Energy (DOE). ORISE is managed by Oak Ridge Associated Universities under DOE contract number DE-AC05-06OR23100. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORISE. We also acknowledge the support of DARPA under contract No. HR0011-06-1-0021 and of NSF under the Graduate Research Fellowship Program.

7. REFERENCES

[1] J. Bates. Virtual reality, art, and entertainment. *Presence: The Journal of Teleoperators and Virtual Environments*, 2(1):133–138, 1992.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] C. L. Isbell, Jr., C. R. Shelton, M. Kearns, S. Singh, and P. Stone. A social reinforcement learning agent. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 377–384, 2001.

[4] M. Kearns, Y. Mansour, and A. Y. Ng. Approximate planning in large POMDPs via reusable trajectories. *Advances in Neural Information Processing Systems*, 12, 2000.

[5] A. Lamstein and M. Mateas. A search-based drama manager. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 2004.

[6] B. Laurel. *Toward the Design of a Computer-Based Interactive Fantasy System*. PhD thesis, Drama department, Ohio State University, 1986.

[7] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML-94)*, pages 157–163, 1994.

[8] B. Magerko. Story representation and interactive drama. In *Proceedings of the First Annual Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-05)*, 2005.

[9] M. Mateas. An Oz-centric review of interactive drama and believable agents. In M. Woodridge and M. Veloso, editors, *AI Today: Recent Trends and Developments. Lecture Notes in AI 1600.* Springer, Berlin, NY, 1999. First appeared in 1997 as Technical Report CMU-CS-97-156, Computer Science Department, Carnegie Mellon University.

[10] M. Mateas and A. Stern. Integrating plot, character, and natural language processing in the interactive drama Façade. In *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE-03)*, 2003.

[11] B. Mott and J. Lester. U-director: A decision-theoretic narrative planning architecture for storytelling environments. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, 2006.

[12] M. J. Nelson and M. Mateas. Search-based drama management in the interactive fiction Anchorhead. In *Proceedings of the First Annual Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-05)*, 2005.

[13] M. J. Nelson, D. L. Roberts, C. L. Isbell, Jr., and M. Mateas. Reinforcement learning for declarative optimization-based drama management. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, 2006.

[14] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-00)*, pages 663–670, 2000.

[15] Z. Rabinovich and J. S. Rosenschein. Multiagent coordination by extended Markov tracking. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05)*, pages 431–438, 2005.

[16] Z. Rabinovich and J. S. Rosenschein. On the response of EMT-based control to interacting targets and models. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, 2006.

[17] M. O. Riedl, A. Stern, and D. Dini. Mixing story and simulation in interactive narrative. In *Proceedings of the Second Annual Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-06)*, 2006.

[18] D. L. Roberts, M. J. Nelson, C. L. Isbell, M. Mateas, and M. L. Littman. Targeting specific distributions of trajectories in MDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, Boston, MA, 2006.

[19] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. Unpublished. URL: http://web.cps.msu.edu/rlr/pub/Tesauro2.html.

[20] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[21] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[22] P. Weyhrauch. *Guiding Interactive Drama*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1997. Technical Report CMU-CS-97-109.

[23] R. M. Young, M. O. Riedl, M. Branly, A. Jhala, R. J. Martin, and C. J. Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1), 2004.