Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence

Planning for Operational Control Systems with Predictable Exogenous Events*

Ronen I. Brafman
Computer Science Dept.
Ben-Gurion Univ. of the Negev, Israel
brafman@cs.bgu.ac.il

Carmel Domshlak
Industrial Eng. and Mgmt.
Technion–Israel Inst. of Technology
dcarmel@ie.technion.ac.il

Yagil Engel and **Zohar Feldman**
IBM - Haifa Research Lab
Haifa, Israel
{yagile,zoharf}@il.ibm.com

Abstract

Various operational control systems (OCS) are naturally modeled as Markov Decision Processes. OCS often enjoy access to predictions of future events that have substantial impact on their operations. For example, reliable forecasts of extreme weather conditions are widely available, and such events can affect typical request patterns for customer response management systems, the flight and service time of airplanes, or the supply and demand patterns for electricity. The space of exogenous events impacting OCS can be very large, prohibiting their modeling within the MDP; moreover, for many of these exogenous events there is no useful predictive, probabilistic model. Realtime predictions, however, possibly with a short lead-time, are often available. In this work we motivate a model which combines offline MDP infinite horizon planning with realtime adjustments given specific predictions of future exogenous events, and suggest a framework in which such predictions are captured and trigger real-time planning problems. We propose a number of variants of existing MDP solution algorithms, adapted to this context, and evaluate them empirically.

Introduction

Operational control systems[1] (OCS) are devised to monitor and control various enterprise-level processes, with the goal of maximizing enterprise-specific performance metrics. These days, OCS is a beating heart of various heavy industries, financial institutions, public service providers, call centers, etc. (Etzion and Niblett 2010). While different OCS employ different degrees of automation, a closer introspection reveals an interesting wide common ground. In most cases, as long as the enterprise is operating in its "normal conditions", its control by the OCS is mostly automatic, carried out by a default policy of action. However, once deviations from the "normal conditions", or *anomalies*, are either detected or predicted, it is common that a human decision maker takes control of the system and changes the policy to accommodate the new situation. Anomalies, such as machine failures, severe weather conditions, and epidemic bursts are detected either by system operators, or through an automatic *event-processing* system. Because the state models of enterprises adopting OCS are complex and because the decisions need to be timely, the outcome of human decisions in face of exogenous events are often sub-optimal, leading to unnecessary high costs for the enterprises. Achieving a higher degree of OCS automation is thus clearly valuable, and this is precisely the agenda of our work here.

At a high level, OCS are naturally modeled as Markov decision processes (MDP), and previous works on MDPs with exogenous events suggested compiling a prior distribution over these events into the probabilistic model of effects of actions performed by the system (Boutilier, Dean, and Hanks 1999). OCS, however, challenge this approach threefold. First, the space of exogenous events and their combinations, which may impact an OCS can be very large, prohibiting their modeling within the MDP. Second, for many of these exogenous events there is no useful predictive, probabilistic model. Finally, even if accurate priors are available, realtime predictions, possibly with short lead-time, make these priors effectively useless. Given an information about ongoing and near-coming events, the model has to be updated, and a good/optimal policy for it needs to be computed. The major computational issue here is that the new policy should be computed in real time, during system operation, and thus there is usually not enough time to solve the new model from scratch.

In this work we introduce a model that combines offline MDP infinite horizon planning with realtime adjustments given specific predictions of exogenous events, and suggest a concrete framework in which such predictions are captured and trigger real-time planning problems. At a high level, the model is based on the characteristic OCS property, whereby exogenous events constitute anomalies, derailing the system from some concrete default mode of operation. Specifically, the model exploits the fact that (i) the MDP representing the system is expected to differ from the default MDP only within a specific time interval affected by a set of exogenous events, and that (ii) most of the actions prescribed by the pre-computed policy for the default mode of operation are likely to remain optimal within the event-affected period. Under this perspective, we suggest modeling predictions of exogenous events as trajectories in a space of different MDPs, each capturing system operation under spe-

*Brafman and Domshlak were partly supported by ISF Grant 1101/07. We thank anonymous reviewers for helpful comments.

[1]We note that the terminology in this area is still evolving, and OCS are called differently in different domains.

cific external conditions. The overall decision process then combines fixed horizon MDPs corresponding to the states of the predicted trajectory, with infinite horizon MDP capturing the default working mode. Beyond formalizing and motivating this model, our main contributions are in providing a new variant of policy iteration for this setting, and empirically evaluating it, along with adaptations of other MDP algorithms, on a call center model.

Operational Control by Example

Before proceeding with formalizing our decision process model for OCS, we wish to provide the reader with a better intuition of daily problems addressed by these systems. We therefore commence by describing two realistic scenarios.

Example 1. *Consider a call center of a large company, such as a communication service provider, or a B2C retailer, or a travel agency. Typically, such a company is doing most of its business online, but needs to provide telephone support for cases that cannot be handled over the web or for users that prefer human interaction. The state of the call center comprises the current availability of representatives, and may also include information regarding the expected distribution of types of calls in the near future. The so called "routing policy" of the call center indicates the type of representative to which a call type is assigned at any given state. Beside that, the company needs to decide, on a daily basis, how many representatives of each type it will need on the next day. An important part of the reality is that some representatives hired for the next days may still not show up.*

Putting the specifics of the call center aside, the example above actually represents a typical enterprise operational control problem of resource management and allocation in face of uncertainty about resource availability and the demand for the enterprise's services. Two kinds of actions are typically available to the system: actions that change resource-allocation policy, and actions that affect the availability of resources for a certain period of time. Considering the demand in the context of Example 1, there are various factors that are known to affect the distribution of calls. For example, if the main business of the company is done through web servers, and a webserver crashes, a surge in the number of calls will soon follow. A crash can in turn be sometimes predicted in advance based on specific patterns of preceding events. Other factors can affect the volume of calls. For instance, travel agencies are affected by terror alerts that typically lead to waves of cancellations, usually a day or two after the alert is published, and often cancellations cannot be done through the website.

Example 2. *A power company controls the production of electric power and its allocation to geographic areas. Similarly to Example 1, given a set of resources (power production in specific plants) and demands (coming from different geographic areas), the overall operational control problem is of resource management to match the set demands. The actions available to the power company correspond to increasing/decreasing production at various costs; for instance, the company can start up diesel generators (approximately ten times more expensive than coal generators). In addition, the company can launch emergency fix to its coal generators (coal generators normally lose some of their production capacity over time, and are fixed according to a regular maintenance schedule). Finally, power companies can also affect consumption through particular pricing mechanisms with industrial consumers; the contract with these consumers defines the timing in which a price increase can be announced.*

Here as well, the ability to predict both production constraints and demand for the service, significantly affects system's capability of acting optimally. Here as well, different events affecting the production and demand, such as weather conditions, can be predicted sufficiently in advance. We note that the palette of OCS applications goes beyond the resource management class of problems; additional examples can be found in Engel and Etzion (2011).

Related Work

Our work is related to other MDP approaches involving online planning. The influential work of (Kearns, Mansour, and Ng 2002) describes an online algorithm for planning in MDPs. This algorithm selects the next action to execute by using sampling to approximate the Q value of each action for the given state by building a sparse look-ahead tree. Its complexity is independent of the number of states, but is exponential in the horizon size. Our setting is similar, in the sense that we, too, need to compute a bounded horizon policy online. However, in our setting the online planning algorithm can exploit the current policy and the current value function in order to be able to respond faster. In general, we believe that sampling methods could play an important role in our setting as well. There are other online methods for MDPs, including replanning (Yoon et al. 2008), which are suitable for goal oriented domains. RTDP (Barto, Bradtke, and Singh 1995) as well can be viewed as a sampling based approach, and because of its bias, it is likely to explore paths near those generated by the current policy.

Our work differs also from the typical way exogenous events are integrated into MDPs (Boutilier, Dean, and Hanks 1999). The typical assumption is that exogenous events are part of the decision process model, whether implicitly (in the action transition matrices) or explicitly (by modeling their effects and the probability of their occurrence). Even in the latter case, these events are compiled away, at which point any standard algorithm can be applied. Our basic assumption is that there is no model that describes the probability of an exogenous event, only a model of the effects of such an event, and hence these events cannot be compiled away. Finally, there is much work that deals with model uncertainty in MDPs, centered around two typical approaches. The first is to cope with uncertainty about the model; for example, by using representations other than probability for transitions (e.g., (Delgado et al. 2009)). The second is to compute policies that are robust to changes (e.g., (Regan and Boutilier 2010)). Both methods are somewhat tangential to our setting which focuses on the exploitation of concrete predictions of exogenous events. In the absence of such an event the model is assumed to be accurate.

Finally, event processing systems that we exploit here are already in wide and growing use in OCS (Etzion and Niblett 2010). Though currently most of these systems do not exhibit predictive capabilities and focus only on detecting certain ongoing events, this is expected to change in the next generation of these systems. Preliminary work in this direction has been done by Wasserkrug, Gal, and Etzion (2005) and Engel and Etzion (2011).

Planning with External Factors

As discussed in the introduction, OCS usually have a default work mode which is applicable most of the time. In our example scenarios, the call center has a usual distribution over incoming calls, and the power company has its usual consumption/production profile. We model this "normal" or "default" dynamics of the world as a Markov decision process $M^0 = \langle \mathcal{S}, A, T^0, R \rangle$ where $\mathcal{S}$ is a set of states, A is a set of actions, $R : \mathcal{S} \times A \rightarrow \Re$ is the reward function, and $T^0 : \mathcal{S} \times A \times \mathcal{S} \rightarrow [0, 1]$ is a transition function with $T^0(s, a, s')$ capturing the probability of achieving state s' by applying action a in state s; for each $(s, a) \in \mathcal{S} \times A$, $\sum_{s' \in \mathcal{S}} T^0(s, a, s') = 1$.

The exogenous events we are concerned with alter the dynamics of the default MDP by changing its transition function. Different events (e.g., heat-wave, equipment failure, or a major televised event) will have different effects on the dynamics of the system, and hence would change the transition function in different ways. We use an external variable $\mathcal{E}$ to capture the value/type of the exogenous effect. Hence, our MDP is more generally described as $M(E) = \langle \mathcal{S}, A, T(E), R \rangle$, where $E \in dom(\mathcal{E})$.[2] We use E_0 to denote the default value of $\mathcal{E}$, and therefore $M^0 = M(E_0)$. The state space, actions, and the reward function are not affected by the exogenous events.

A basic assumption we make is that M^0 has been solved, possibly offline, and its optimal (infinite horizon, accumulated reward with discount factor γ) value function v^0 is known. This reflects the typical case, in which an OCS has a default operation policy π^0 in use. If $\mathcal{E}$ has a small domain, we might be able to solve $M(E)$ for all possible values ahead of time. However, as will become clear below, the optimal policy for $M(E)$ is not an optimal reaction to an event E because of the transient nature of the changes induced by exogenous events. But on a more fundamental level, in practice the set of potential exogenous events is likely to be huge, so presolving the relevant MDPs is not an option. In fact, in many natural scenarios, we expect that the influence of the exogenous effect on the transition function will be *part* of the prediction itself. The option of compiling the model of exogenous events into the MDP is at least as problematic, both because of the size issue, but more fundamentally, because we may have no good predictive model for some of these events.

[2]We do not consider the question of how this mapping from $\mathcal{E}$ to MDPs is represented internally. A reasonable representation could be a Dynamic Bayesian Network (see Boutilier, Dean, and Hanks (1999)) in which the exogenous variable(s) appear in the first (pre-action) layer, but not in the post-action layer.

Formally, a prediction takes the form of a sequence of pairs $((E_0, \tau_0), (E_1, \tau_1), \ldots, (E_m, \tau_m))$, each $E_i \in dom(\mathcal{E})$ capturing a specific exogenous event, and τ_i capturing the number of time steps for which the system is expected to be governed by the transition model $T(E_i)$. We will refer to such predictions as *trajectories*, reflecting their correspondence to trajectories in the space of possible MDPs $\{M(E) \mid E \in dom(\mathcal{E})\}$. While $\tau_0 \geq 0$, for $i > 0$, we have $\tau_i > 0$; the former exception allows for modeling predictions of immediate change from the default value E_0. A useful notation for the absolute time in which the dynamics of the system shifts from $T(E_{i-1})$ is $t_i = \sum_{j=0}^{i-1} \tau_j$, $i \in \{0, \ldots, m+1\}$. Our basic assumption, inspired by the natural usages of OCS described earlier, is that at time t_{m+1} the system returns to its normal, default mode. This captures the idea that the predictions we consider stem from exogenous events with a transient effect on the system.[3]

We can now formulate the basic technical question posed by this paper: *How can an OCS quickly adapt its policy given a prediction as captured by a trajectory.*

Algorithms for Realtime Policy Adjustment

Once a prediction $((E_i, \tau_i))_{i=0}^m$ is received, behaving according to M^0 may no longer be optimal because it no longer reflects the expected dynamics of the world. Moreover, because the transition function is no longer stationary, we do not expect the optimal policy to be stationary either, and the choice of action must depend on the time-step as well, to reflect the dependency of value on the time. For example, if we get a warning that the frequency of calls will increase significantly 3 days from now, and it takes just one day to recruit additional representatives, the value of "regular states" only begins to deteriorate 2 days from now.

This setting corresponds to a series of $m + 1$ finite horizon MDPs defined by the trajectory, followed by an infinite horizon MDP defined by M^0. For notational convenience, we consider the new setting as a meta-MDP whose states are denoted by s_t, $t \geq 0$, and $v(s_t)$ denotes the value function of its optimal policy. We also use T^t to denote the transition probabilities governing the MDP at step t, that is $T^t = T(E_i)$ if $t_i \leq t < t_{i+1}$ for some $0 \leq i \leq m$, and $T^t = T(E_0)$ if $t \geq t_{m+1}$. In principle, the value function of the optimal policy for this meta-MDP can be computed in time $O(|\mathcal{S}|^2 \cdot |A| \cdot t_{m+1})$ via backward induction, by propagating backward the (known) values of the states in the infinite-horizon MDP M^0 terminating this trajectory, that is,

$$v(s_t) = \begin{cases} v^0(s), & t \geq t_{m+1} \\ \gamma \max_a \left[R(s, a) + \sum_{s' \in S} T^{t+1}(s, a, s') v(s'_{t+1}) \right], & t < t_{m+1} \end{cases} \quad (1)$$

In many applications, however, and in particular in OCS, we do not expect solving the entire meta-MDP via Eq. 1 to be feasible because the state space is large while time to react is short. In order not to be left without a response, in

[3]Extension of the methods presented in this paper to probabilistic trajectories is conceptually straightforward.

such cases we should adopt this or another anytime computation, quickly generating a good policy without exploring the entire state space. Since in our problem we always have a concrete initial state, which we denote by $\hat{s}$, search-based algorithms appear the most promising alternative here. Because v^0 is known, we can treat all states at time t_{m+1} as leaf nodes in our search tree; while the size of the complete search tree is exponential in t_{m+1}, when the time horizon is short and the state space is large, we expect this number to be substantially smaller than $|\mathcal{S}|^2|A|t_{m+1}$. By using heuristic search techniques, we expect to make further gains, exploring only a small portion of this search tree.

In what follows we elaborate on several possible adaptations of existing MDP solution algorithms while attempting to exploit the properties and address the needs of the OCS-style problems. In particular, we make use of an assumption that exogenous events cause only unfavorable changes in the system dynamics.

Assumption 1. *For any state $s \in \mathcal{S}$, and any time step $t \in \{0, \dots, t_{m+1}\}$, we have that $v(s_t) \leq v^0(s)$.*

While in general this is, of course, a heuristic assumption, it very often holds in the systems we are interested in. For instance, in resource management problems, the external events affecting the operational decisions either decrease available resources or increase demand. Note that, if this assumption holds, then the default-mode value function v^0 provides us with *admissible* value estimates for the time-stamped states of our meta-MDP.

Search Based on AO*

AO* is arguably the most prominent framework for planning by search in MDPs (Hansen and Zilberstein 2001). When used with an admissible heuristic, AO* is guaranteed to converge to an optimal solution. At high level, AO* maintains a current best solution graph, from which it selects a tip node to expand. The expansion results in a value update, using the heuristic values of the newly revealed nodes. The update may lead to change of policy and hence a change of current best solution graph. The main parameter for AO* is a selection procedure for which tip node in the current optimal solution graph shall be expanded. The selection we have used in our evaluation was to expand the node associated with the time-stamped state having the highest probability to be reached from $\hat{s}$. Our motivation being that, as this is an online setting, we would like to improve the most likely trajectory of the policy induced by the current solution graph. However, as we discuss below, this selection principle is not free of pitfalls.

Given a meta-MDP, AO* will start by expanding the reachability tree of π^0 for t_1 time steps. This follows from the fact that the heuristic function is the optimal value function of M^0 and that during the first t_1 steps, the dynamics of the system corresponds to its transition function T^0. Thus, for each time-stamped state s_t with $t < t_1$, one of its greedily optimal actions will correspond to $\pi^0(s)$. However, once we reach depth t_1, the transition probabilities change; this results in *updated heuristic values* to s_{t_1} and its predecessors. After the update, $\pi^0(s)$ is not necessarily optimal. In the context of action execution, in parallel to the computation, we now have a dilemma: should we keep on following the default policy π^0, even though its heuristic value is now suboptimal, or switch to follow the new greedy policy, which follows the best *updated* heuristic values? It appears that the latter can be dangerous. Considering the call center example, assume that we have an action reducing the staff of representatives, and that this action is not optimal for $\hat{s}$ because all of the current staff is really needed. Now assume that the first external event is expected to increase demand. The current policy leads to a value lower than $v^0(\hat{s})$, because the current staff is insufficient, and this value is perhaps even lower than the value we had under M^0 for the action of dismissing staff members. Hence, if we follow the updated heuristic, we might release staff members and end up in a worse situation.

The example above illustrates our rationale behind not following the updated heuristic before we have a robust information about the actions it proposes to follow. This reveals a drawback of AO* in our online settings: if we keep on following the old policy, and let the algorithm continue exploring the prospects of changing the policy close to the initial state (where the meta-MDP states have higher probability to be reached), then a dominant part of our computation will turn out to be redundant: most of the time-stamped states explored by the search may no longer be reachable after the next action is performed by the system. Even if we select a different node expansion policy, AO*'s breadth-first flavor is going to prevent us from exploring more deeply the implication of a change, and therefore, we may not have high confidence in its intermediate greedy choice.

Search Based on Branch and Bound

A popular search scheme that allows for exploring new options more deeply is that of depth-first (or AND/OR) branch and bound (DFB&B) algorithm (Marinescu and Dechter 2005). The specialization to our setting involves using our heuristic value v^0 as the upper bound, using the transition probabilities appropriate to the current time level in the search tree, and merging paths as in AO*. Informally, DFB&B works in our setting as follows: expand the plan tree from current state according to π^0, and evaluate it from its leaves upwards; at each node check whether there exists an alternative action whose value under M^0 (assuming π^0 is performed in the rest of the states) is better than the updated value of the current action. If there is, update π^0, and expand the tree again from that state (to the last time step of the horizon relative to the initial state) and start the backward propagation again.

Biasing towards the current policy as our first branch, we tend to solve smaller sub-trees closer to the end of the trajectory. This ensures that the computation made will be relevant much longer. Importantly, if we must execute actions before computation is complete, the execution and the computation *will necessarily meet along the tree*.

There is no free lunch, however. The Achilles heel of DFB&B is its exhaustive search in the state space, alleviated only by the prospects of pruning due to the upper bound. In particular, DFB&B initially must try all possible policy improvements very close to the end of the time horizon. In the OCS settings, however, it is unlikely that significant improvements can be found at steps which are close to the end of the trajectory; usually, the policy should change earlier.

Lazy Policy Iteration

Having in mind the relative pros and cons of AO* and DFB&B, we now propose a different algorithm for online solving our meta-MDPs that tries to accommodate the best of both worlds: bias towards modifying the current policy at early time steps, as well as adaptation to the outcomes of the in-parallel execution. This algorithm can be seen as a variant of the modified policy iteration scheme; we call it *lazy policy iteration (LPI)* because it performs the value determination step of policy iteration only very selectively. At high level, at each policy iteration the algorithm tries to improve the current policy only for a single state-action pair $(\bar{s}_t, \bar{a})$. LPI is a form of asynchronous modified policy iteration (Singh and Gullapalli 1993); however, it prioritizes its value backups heuristically, and takes advantage of the knowledge of current state and model-change horizon.

Figure 1 provides a pseudo-code description of the algorithm. The algorithm incrementally updates a current policy π that is initially set to π^0, as well as a value function v, initialized to v^0, and kept up to date with π for any meta-MDP state generated so far. The algorithm also maintains a function $\hat{h}(s_t, a) = \sum_{s'} T^t(s_t, a, s')v(s'_{t+1})$. After the initial expansion of the reachability graph of π, a candidate pair $(\bar{s}_t, \bar{a})$ is chosen to maximize

$$\delta^* = \max_{(s,t,a)} \left[p(s_t) \cdot (\hat{h}(s_t, a) - v(s_t))\right], \qquad (2)$$

where $p(s_t)$ denotes the probability of reaching s from $\hat{s}$ in t steps. The selected state-action pair is then passed to the function EXPAND where, for the sake of readability, it is referred by $(\bar{s}_{t^*}, \bar{a})$.

The function EXPAND defines a candidate policy

$$\pi'(s_t) = \begin{cases} \bar{a} & s_t = \bar{s}_{t^*} \\ \pi(s_t) & \text{otherwise,} \end{cases} \qquad (3)$$

and computes its value, captured by the Q function:

$$Q(\bar{s}_{t^*}, \bar{a}) = R(\bar{s}, \bar{a}) + \gamma \sum_{s'} T(\bar{s}, \bar{a}, s')v(s'_{t+1}) \qquad (4)$$

This is done by generating the reachability graph of π' from state $\bar{s}_{t^*}$ up to the time step t_{m+1}, and then computing/updating bottom-up the value $v(s_t)$ for all the generated states s_t, along with the estimates $\hat{h}(s_t, a)$ for all actions $a \neq \pi'(s_t)$. At a more detailed level, there is no need for EXPAND to actually generate and recompute the value of states s_t that have already been generated before. However, the probabilities $p(s_t)$ computed along the way should be computed for all, both previously and newly generated, states reachable from $\bar{s}_{t^*}$ along π up to time t_{m+1}.

Returning now to the main algorithm, if $Q(\bar{s}_t, \bar{a})$ computed by EXPAND for the candidate pair $(\bar{s}_t, \bar{a})$ turns out to be higher than $v(\bar{s}_t)$, then π' replaces π as our current policy, and v is adjusted to capture the value of π' using a local

algorithm LAZYPOLICYITERATION
  input: current state $\hat{s}$, policy π^0, value v^0, trajectory τ
  output: a policy π for meta-MDP induced by τ
foreach meta-MDP state s_t
  schematically set $\pi(s_t) = \pi^0(s)$ and $v(s_t) = v^0(s)$
EXPAND$(\hat{s}_0, \pi^0(\hat{s}), \pi, \tau)$
while time permits **do**
  $\delta^* = \max_{(s,t,a)}[p(s_t) \cdot (\hat{h}(s_t, a) - v(s_t))]$
  $(\bar{s}_t, \bar{a}) = \arg\max_{(s,a,t)}[p(s_t) \cdot (\hat{h}(s_t, a) - v(s_t))]$
  if δ^* is small enough
    return π
  EXPAND$(\bar{s}_t, \bar{a}, \pi, \tau)$
  if $Q(\bar{s}_t, \bar{a}) > v(\bar{s}_t)$
    $\pi(\bar{s}_t) = \bar{a}$
    $v(\bar{s}_t) = Q(\bar{s}_t, \bar{a})$
    foreach node s_t on a backward path from $\bar{s}_t$ to $\hat{s}$
      update $v(s_t)$ according to Eq. (1)
return π

procedure EXPAND (state $\bar{s}_{t^*}$, action $\bar{a}$, policy π, trajectory τ)
define π' according to Eq. (3)
for $t := t^* + 1$ **to** t_{m+1}
  generate all states s_t reachable from $\bar{s}_{t^*}$ along π', and compute $p(s_t)$
for $t := t_{m+1} - 1$ **down to** t^*
  foreach state s_t reachable from $\bar{s}_{t^*}$ along π'
    $v(s_t) = R(s, \pi'(s_t)) + \gamma \sum_{s'} T^t(s, \pi'(s_t), s')v(s'_{t+1})$
    foreach action $a \neq \pi'(s_t)$
      $\hat{h}(s_t, a) = \sum_{s'} T^t(s_t, a, s')v(s'_{t+1})$
$Q(\bar{s}_{t^*}, \bar{a}) = R(\bar{s}, \bar{a}) + \gamma \sum_{s'} T^t(\bar{s}, \bar{a}, s')v(s'_{t+1})$

Figure 1: Lazy Policy Iteration.

application of Eq. (1). Such iterative improvement of the current policy is performed until either time is up or value has ϵ-converged. Note that if in parallel the runtime system executes an action (instructed by the current policy π), the algorithm continues with $\hat{s}$ updated to the new current state.

A simple way to understand lazy policy iteration is as follows. The standard policy iteration algorithm iteratively performs two steps: value determination, and policy improvement. Value determination computes the value of the current policy for all states, while policy improvement improves the policy in a greedy manner, by switching the policy on some state(s) to action(s) whose Q value is higher than its current value. Lazy policy iteration performs value determination only when it must: Rather than computing the Q values for all state-action pairs, it does so only for a single, most promising pair. That single Q value is computed by expanding the state/time subgraph below the selected pair. If this Q value is indeed an improvement, it changes the policy.

Lazy policy iteration aims at avoiding the pitfalls of the previous algorithms. The Q-value update selection procedure is biased towards significant states, and the algorithm does not perform a full depth-first search of their sub-trees, but only of the particular portions of these sub-trees, corresponding to continuing with the current policy. It is guided to explore particular actions, rather than those that are too low in the tree (as DFB&B) or those that will never be followed if execution happens at the same time according to the default policy (as AO*).

Experimental Results

We tested the performance of lazy policy iteration, AO*, and backwards induction on a call center model, inspired by real call center data available at IBM Research. The model includes three types of queries, and two types of agents, such that each agent can handle two of the three types of queries. This model is often referred to as W-design, stemming from the shape formed by its schematic representation (Garnett and Mandelbaum 2001). There is a fixed size of queue per query type; calls are assigned to an appropriate agent if one is available, otherwise wait in the queue, or blocked if the queue is full. Calls of query type i arrive according to a *Poisson Process* with rate ρ_i (the expected number of call arrivals per unit of time is ρ_i), and the time a query of type i is served by an agent of type j is modeled as an exponential distribution with parameter σ_{ij}. The reward function is the negation of two cost factors: a waiting cost incurred by each queued call per time unit, and a cost for each call that is blocked. We tested the algorithms on a call center with 25 agents of each type, and queue capacities of 30. The service rates depend merely on the agent type, so that the state of the system can be described by the number of calls in each of the queues, and the number of busy agents from each type. This settings yield a state space of about 100,000 states.

In the scenario we tested, the initial distribution of calls is divided evenly between the query types. A trajectory predicts that after t_1 time steps, the distribution changes; under the new distribution, ρ_1 is five times the current value of ρ_2 and ρ_3, and the latter ones remain unchanged. The system returns to normal after additional t_2 time steps. The policy change we expect to occur when such prediction is received is to divert more calls from the middle queue to the agent whose other queue is type 3, as he is expected to be less busy. We report the performance of the algorithms, ran to optimality, as a function of the time horizon, which is $t_1 + t_2$. The results (Figure 2) confirm that approaches based on locality outperform backward induction for $t_{m+1} < 75$ (lazy policy iteration), and $t_{m+1} < 30$ (AO*). Furthermore, lazy policy iteration significantly outperforms AO* (irrespective of its *anytime* performance advantage). We note that the value of $v^0(\hat{s})$ deteriorates as we increase t_2; the value of the optimal online policy (on $\hat{s}_0$) is about 20% higher than $v^0(\hat{s})$.

In a second set of experiments (Figure 3), we compared the performance of lazy policy iteration and backwards induction, with $t_1 + t_2 = 50$, and varying the size of the state space As expected, the dependence of backwards induction on the size of the state space is polynomial. In contrast, with a fixed time horizon, lazy policy iteration hardly exhibits any dependency on the size of the state space.

Conclusions

In this work we take a step towards practical application of decision-theoretic planning in daily business setting. We present a model in which exogenous events are predicted, usually with a short lead-time. We consider several approaches to compute a temporary change of policy; our lazy policy iteration (LPI) scheme facilitates interleaving computation with execution. Empirical evaluation on a call center

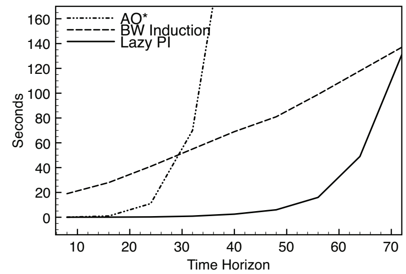

Figure 2: Performance as a function of time horizon.

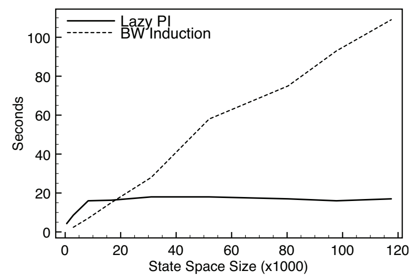

Figure 3: Performance as a function of number of states.

model finds LPI significantly more efficient for short horizons than the other approaches we considered.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Inteligence* 72(1):81–138.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of AI Research* 11:1–94.

Delgado, K. V.; Sanner, S.; de Barros, L. N.; and Cozman, F. G. 2009. Efficient solutions to factored MDPs with imprecise transition probabilities. In *ICAPS*.

Engel, Y., and Etzion, O. 2011. Towards proactive event-driven computing. In *DEBS*.

Etzion, O., and Niblett, P. 2010. *Event Processing in Action*. Manning Publications.

Garnett, O., and Mandelbaum, A. 2001. An introduction to skills-based routing and its operational complexities. *Teaching note, Technion*.

Hansen, E., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning* 49(2-3):193–208.

Marinescu, R., and Dechter, R. 2005. AND/OR branch-and-bound for graphical models. In *IJCAI*.

Regan, K., and Boutilier, C. 2010. Robust policy computation in reward-uncertain MDPs using nondominated policies. In *AAAI*.

Singh, S. P., and Gullapalli, V. 1993. Asynchronous modified policy iteration with single-sided updates. *unpublished*.

Wasserkrug, S.; Gal, A.; and Etzion, O. 2005. A model for reasoning with uncertain rules in event composition. In *UAI*.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*.