2011 American Control Conference
on O'Farrell Street, San Francisco, CA, USA
June 29 - July 01, 2011

Computing Policies and Performance Bounds for Deterministic Dynamic Programs Using Mixed Integer Programming

Randy Cogill
Dept. of Systems and Information Engineering
University of Virginia
rcogill@virginia.edu

Haitham Hindi
Intelligent Systems Lab
Palo Alto Research Center
hhindi@parc.com

Abstract—**In this paper we present a mixed integer programming approach to deterministic dynamic programming. We consider the problem of computing a policy that maximizes the total discounted reward earned over an infinite time horizon. While problems of this form are difficult in general, suboptimal solutions and performance bounds can be computed by approximating the dynamic programming value function. Here we provide a linear programming-based method for approximating the value function, and show how suboptimal policies can be computed through repeated solution of mixed integer programs that directly utilize this approximation. We have applied this approach to problems with states described by binary vectors with dimension as large as several hundred. Although the number of distinct states associated with such a problem is extremely large, we are able to obtain suboptimal policies with surprisingly tight performance guarantees. We illustrate the application of this method on a class of infinite horizon job shop scheduling problems.**

I. Introduction

When modeling static combinatorial optimization problems, it is often possible to find compact representations of problems as binary integer programs. The difficulty in solving these problems lies in the fact that the space of feasible solutions to such a problem is generally too large to enumerate, and there are no algorithms that are guaranteed to search this space efficiently. However, despite the fact that there are no algorithms guaranteed to efficiently solve all problem instances, large problem instances are still routinely solved in practice. The ability to solve large problem instances is primarily enabled by search strategies that exploit the connection between binary integer programs and their linear program relaxations. In particular, linear program relaxations can provide bounds on achievable performance, and provide a key building block to many of the effective algorithms for computing globally optimal solutions such as branch-and-bound and cutting plane methods.

The goal of this work is to bring the advantages provided by integer programming models to *dynamic* optimization problems. Dynamic programming models of realistic systems often require models with a high-dimensional state, and hence suffer from an explosive growth in the size of their state-space. When this is the case, algorithms that require directly enumerating the state space are not practical.

Here we present an integer programming approach for solving deterministic dynamic programs. The advantage of our proposed class of models is that we can often represent systems with a high dimensional state without directly enumerating all states. To solve the resulting dynamic optimization problems, we can use an approximation method that implicitly approximates a dynamic programming value function. Value function approximation is a well known approach to computing suboptimal policies for complex dynamic problems [16], [4], [13]. The control theory community is also increasingly embracing techniques related to approximate dynamic programming for control of complex systems, as evidenced by a number of recent papers on the subject (for example [20], [10], [11], [5], to name a few). Ideas similar to those presented in this paper have also been developed in the recent literature on model-predictive control [3], [14], [2].

In the models that we consider, the state of the system can be described by a binary vector, the evolution of the system can be described by a set of linear constraints, and the per-period cost or reward can be expressed as a linear function of the binary state vector. The main result of this paper is a linear programming-based method for approximating the dynamic programming value function of these problems. This linear programming approximation can be used to generate a control sequence by solving moderately-sized mixed-integer linear programs in each time period. By adopting this algorithm, we are able to produce solutions to dynamic programming problems with very large state spaces.

Although the algorithms presented in this paper may not generally produce optimal solutions, this approach exhibits three properties that we believe make this a well

This research was supported by the Xerox Foundation, University Affairs Committee (UAC) Grant: HE 1571-2008.

justified heuristic. In particular:

- On an instance-by-instance basis we can determine how far from optimal a computed solution is. When applying the approach of this paper to a problem, we obtain a suboptimal solution and an upper bound on the reward achievable by an optimal solution. If the reward achieved by the suboptimal solution is close to the upper bound, we have a guarantee that the computed solution is close to optimal.
- If we are unsatisfied with the optimality guarantee produced by the algorithm, we can compute a sharper solution and guarantee at the expense of increased computation. That is, we can compute solutions by solving larger mixed-integer linear programs, which provably closes the gap between the reward achieved by the computed solution and the upper bound on achievable performance.
- We have demonstrated practical applicability on a class of nontrivial problems. Here we provide the results numerical experiments where the methods presented in this paper were applied to a class of dynamic job shop scheduling problems. In these experiments, we observe that reasonable solutions are computed in a reasonable amount of time.

We believe that, in the same way that binary integer programs provide a very general modeling framework for static combinatorial optimization problems, the class of models that are discussed in this paper provide a general modeling framework for deterministic, dynamic combinatorial optimization problems.

II. Problem formulation

Here we present a class of integer programming-based models for deterministic dynamic programs. The three defining characteristics of models considered in this paper are (a) the state-space is a subset of the set of n-dimensional binary vectors (b) the per-period reward function is linear in the state, and (c) the set of feasible state transitions is specified implicitly by a set of linear constraints. Specifically, problem instances are modeled by:

- A binary-valued state vector $x_t \in \{0,1\}^n$
- A reward of $r^T x_t$ earned in each time period
- State transitions specified implicitly by linear constraints. Given state x_t in period t, we can transition to *any* binary x_{t+1} satisfying

$$A_1 x_{t+1} - A_2 x_t \leq b$$

In this formulation, the control action is specified implicitly by the choice of state x_{t+1} given the current state x_t. Furthermore, throughout this paper we assume that there exists at least one feasible sequence $x_1, x_2, \ldots$ associated with each feasible initial state x_0. As an example, we model a job-shop scheduling problem in this framework in Section IV.

Within this framework, one can consider various measures of the reward earned over time. In this paper we consider the total discounted reward achieved over an infinite time horizon. Given a *discount factor* $\alpha \in [0,1)$, the total discounted reward associated with the sequence of states $x_1, \ldots$ is $\sum_{t=0}^{\infty} \alpha^t r^T x_t$. For a given initial state x_0, let $\mathcal{C}(x_0)$ denote the set of feasible sequences $x_0, x_1, x_2, \ldots$ with initial state x_0. We denote the least upper bound on achievable total discounted reward, the *value function*, as

$$V(x_0) = \sup_{x_0, x_1, \ldots \in \mathcal{C}(x_0)} \left\{ \sum_{t=0}^{\infty} \alpha^t r^T x_t \right\} \tag{1}$$

If it is possible to show further that there exists a sequence in $\mathcal{C}(x_0)$ that achieves this supremum.

We note that the problem we address in this paper can also be viewed as a semi-infinite integer program. Semi-infinite programming models have been long been used for representing dynamic optimization problems [1], [8]. In our approach, we decouple this problem over time by using an approximation of the dynamic programming value function. One paper that is quite similar in spirit to the work we present here is [17], where piecewise-linear value function approximations are used to decouple large mathematical programming models of dynamic optimization problems. Similar concepts have also appeared in the AI planning literature, where the power of compact integer programming representations has been noted. In [19] it is shown shown, quite surprisingly, that integer programming algorithms applied to compact integer programming formulations of planning problems can outperform specialized planning algorithms on numerous benchmark problems.

III. Background and Notation

In principle, optimal solutions to the problems described in the previous section can be solved using dynamic programming. That is, the dynamic programming value function V satisfies *Bellman's equation*,

$$V(x) = \max_{y \in \mathcal{F}_1(x)} \{ r^T x + \alpha V(y) \}$$

for all states x. Here we use $\mathcal{F}_1(x)$ to denote the set of states that can feasibly follow x,

$$\mathcal{F}_1(x) = \{ y \in \{0,1\}^n \mid A_1 y \leq b + A_2 x \} \tag{2}$$

$V(x_0)$	The exact value function for the infinite horizon, discounted reward dynamic program.
$\widetilde{V}(x_0)$	An upper bound on $V(x_0)$.
$U_N(x_0)$	The infinite horizon, discounted reward earned by an N-step lookahead policy that uses $\widetilde{V}(x_0)$ as an approximate value function.
$V_N(x_0)$	The optimal N-horizon, discounted reward.
$\widetilde{V}_N(x_0)$	An upper bound on $V_N(x_0)$. We obtain $\widetilde{V}(x_0)$ as the limit of $\widetilde{V}_N(x_0)$ as $N \to \infty$.

TABLE I
QUANTITIES RELATED TO THE OPTIMAL DYNAMIC PROGRAMMING VALUE FUNCTION

Using this notation, an optimal sequence $x_0, x_1, x_2, \ldots$ can be obtained as

$$x_{t+1} = \underset{y \in \mathcal{F}_1(x_t)}{\operatorname{argmax}} \left\{ r^T x_t + \alpha V(y) \right\}$$

for all $t \geq 0$. The challenge of applying dynamic programming lies in solving Bellman's equation to find the value function. More often than not, the value function does not have any special structure, and must be described by a table specifying $V(x)$ for each individual value of x. In many problems the state space can be astronomically large, making this approach intractable.

If we have an approximation of the value function, call it $\widetilde{V}$, we can use this approximation as a surrogate for V and compute a sequence of states according to

$$x_{t+1} = \underset{y \in \mathcal{F}_1(x_t)}{\operatorname{argmax}} \left\{ r^T x_t + \alpha \widetilde{V}(y) \right\}$$

When $\widetilde{V}$ closely approximates V, the sequence of states computed from (4) is guaranteed to be close to optimal, as will be shown below. More generally, given an x_0 we can consider the problem

$$\begin{aligned} \text{maximize: } & \textstyle\sum_{t=0}^{N-1} \alpha^t r^T x_t + \alpha^N \widetilde{V}(x_N) \\ \text{subject to: } & A_1 x_{t+1} - A_2 x_t \leq b \text{ for } t = 0, \ldots, N-1 \\ & x_t \in \{0,1\}^n \text{ for } t = 1, \ldots, N \end{aligned} \tag{3}$$

Let $\mathcal{F}_N(x_0)$ denote the set of $(x_1, \ldots, x_N)$ in the feasible set of (3) for a given x_0. So, more generally, we can compute a sequence of states $x_1, x_2, x_3, \ldots$ as

$$x_{t+1} = \underset{\substack{y_1 \\ (y_1, \ldots, y_N) \in \mathcal{F}_N(x_t)}}{\operatorname{argmax}} \left\{ r^T x_t + \alpha r^T y_1 + \cdots + \alpha^N \widetilde{V}(y_N) \right\} \tag{4}$$

for all $t \geq 0$. This method of generating a sequence of states is known as an *N-step lookahead policy* [4]. It is known that the sequence generated by this policy can be made close to optimal by either finding $\widetilde{V}$ close to V, or by choosing large N. This claim is precisely characterized in the lemma below, which is proved in the online appendix [6], and will appear in an upcoming journal version of this paper. We note that this lemma is essentially equivalent to the standard proof of convergence of the value iteration algorithm for infinite horizon, discounted problems [4]. We present this lemma here to keep our treatment self contained, and express this result in the notation used in this paper.

Lemma 1. *Suppose $\widetilde{V}$ is a function satisfying $\widetilde{V}(x) \geq V(x)$ for all x. For a given x_0, let $x_1, x_2, x_3, \ldots$ be the sequence of states obtained as*

$$x_{t+1} = \underset{\substack{y_1 \\ (y_1, \ldots, y_N) \in \mathcal{F}_N(x_t)}}{\operatorname{argmax}} \left\{ r^T x_t + \alpha r^T y_1 + \cdots + \alpha^N \widetilde{V}(y_N) \right\}$$

for all $t \geq 0$. Also, let

$$U_N(x_0) = \sum_{t=0}^{\infty} \alpha^t r^T x_t$$

be the total discounted reward achieved by this sequence. $U_N(x_0)$ satisfies

$$V(x_0) - U_N(x_0) \leq \frac{\alpha^N}{1-\alpha} \max_x \{ \widetilde{V}(x) - V(x) \}$$

This lemma provides the main justification for the method presented in Section V. There we provide a linear programming-based approach for implicitly constructing a value function approximation $\widetilde{V}$. If this $\widetilde{V}$ closely approximates V, then the achieved reward $U_N(x_0)$ is close to optimal. Moreover, the $\widetilde{V}(x)$ we compute is an upper bound on $V(x)$ for all x. By comparing $\widetilde{V}(x_0)$ with the reward achieved by a suboptimal sequence, we can determine how far from optimal this sequence is. If we are unsatisfied with the suboptimality gap of a computed sequence, we can increase N (at the cost of increased computation) to obtain a new sequence with a tighter suboptimality gap.

We will conclude this section with a summary of the notation used throughout this paper to represent key quantities. Here we introduce several quantities related to the optimal dynamic programming value function V. These quantities are summarized in Table III.

IV. Illustrative Example: Optimal Job Shop Scheduling

As one of example of the deterministic dynamic programs discussed in this paper, we will consider a class of infinite horizon job shop scheduling problems [12]. In these problems, the goal is to schedule various jobs on a collection of shared machines, where each job has distinct processing requirements. The goal is to maximize some measure of the total value of all jobs completed over an infinite time horizon. Infinite horizon job shop problems present an ideal class of applications for the methods in this paper, since the dynamic evolution of these systems are easily described by linear inequalities. In fact, we have been applying concepts similar to this in this paper to more complex forms of job shop scheduling problems that also include a routing component. Prior work on these problems can be found in [7], [9], [15], [18].

Our infinite horizon job shop scheduling problem is described as follows. An instance of this problem is specified by M machines and J job types. The goal is to process a collection of jobs in a way that optimally utilizes the available machines. Specifically, completing a job of type j earns a reward of ρ_j, and the goal is to maximize the total discounted reward earned over an infinite horizon. The complicating factor is that each job type requires processing on a sequence of multiple machines, and each machine can only process a single job in each time period. This infinite horizon formulation of the job shop problem is somewhat unconventional, but provides a good fit for the applications we aim to apply this approach to [7], [9], [15]. In fact, the discounted cost criterion is a particularly suitable choice for a cost criterion since we value both completing jobs quickly *and* maximizing the per-period output of the system.

As an example, consider an instance that has four machines and two job types. Jobs of type 1 must be processed on machine 1 for three time units, then on machine 4 for two time units, then finally on machine 2 for 1 time unit. We denote this sequence of processing requirements as $\mathbf{1}(3), \mathbf{4}(2), \mathbf{2}(1)$, where the numbers in bold indicate machines and the numbers in parentheses indicate required processing times. The second job has processing requirements $\mathbf{2}(5), \mathbf{1}(2), \mathbf{4}(1), \mathbf{3}(1)$. Both jobs earn a reward of $\rho_i = 1$ upon completion. Our goal is to schedule the processing of jobs as to maximize the total discounted with reward achieved over an infinite time horizon. In order to maximize this reward, we can control when jobs are introduced to the system, which types of jobs are introduced to the system, and the flow of jobs through the system. To control the flow of jobs, we can hold some jobs idle at machines if this is required to avoid sending more than one job to a machine.

When modeling a job shop problem as a deterministic dynamic program, we must find an appropriate representation of its state. The flow of a job through the system can be modeled on a directed graph. To show this in terms of a concrete example, consider the jobs of type 1 in the example shown previously. The graph associated with this job is illustrated in Figure 1. In terms of this graph, instances of the job start at the vertex marked s and must cross three edges corresponding to processing on machine 1, two edges corresponding to processing on machine 4, and one edge corresponding to processing on machine 2 before exiting at the vertex marked f. The state of a job instance can be represented by the most recently traversed edge. In each time period, the edge corresponding to the state of a job instance must transition to an edge neighboring the current state. Note that the graph contains two self loops, allowing jobs to remain idle temporarily before passing from one machine to the next.

Let $\mathcal{G}_j = (\mathcal{V}_j, \mathcal{E}_j)$ be the graph associated with job j. We can represent the state of all instances of job j by a collection of binary variables $x_{j,e,t}$. Here, $x_{j,e,t} = 1$ if edge e was the most recently traversed edge of an instance of job j at time t, and $x_{j,e,t} = 0$ otherwise. Let $\mathcal{O}(v)$ be the set of edges outgoing from vertex v, and $\mathcal{I}(v)$ be the set of edges outgoing from vertex v. Note that the graphs corresponding to jobs have a vertex s with $\mathcal{I}(s) = \emptyset$ and a vertex f with $\mathcal{O}(f) = \emptyset$.

The evolution of the system can be modeled by linear constraints. The flow of instances of job j through $\mathcal{G}_j$ is captured by the constraint

$$\sum_{e \in \mathcal{O}(v)} x_{j,e,t+1} \leq \sum_{e \in \mathcal{I}(v)} x_{j,e,t} \tag{5}$$

for all $v \in \mathcal{V}_j \setminus \{s, f\}$ and all $t \geq 0$. An inequality is used in this constraint, allowing instances of a job to be cancelled prior to completion. Allowing cancellation ensures that a feasible sequence exists for all initial states, a requirement discussed in Section II.

Also, recall that each machine may only process one job at any given time. This can also be modeled as a linear constraint. Let $\mathcal{M}_i$ be the set of (j, e) such that edge $e \in \mathcal{E}_j$ corresponds to processing on machine i. The constraint is then

$$\sum_{(j,e) \in \mathcal{M}_i} x_{j,e,t+1} \leq 1 \tag{6}$$

for all $i = 1, \ldots, M$ and all $t \geq 0$. Note that the constraints (5) and (6) allow for multiple active instances

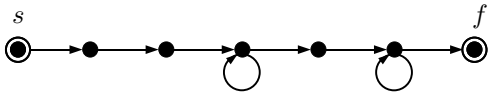

Fig. 1. Graph representing the flow of job $\mathbf{1}(3), \mathbf{4}(2), \mathbf{2}(1)$.

of job j, as long as these instances do not occupy the same machine at the same time.

Finally, a reward of ρ_j is received upon completion of an instance of job j. The reward earned in time period t is given by

$$r^T x_t = \sum_{(j,e)} r_{j,e} x_{j,e,t},$$

where $r_{j,e} = \rho_j$ if $e \in \mathcal{I}(f)$ (*i.e.*, edge e corresponds to the final processing stage of job j), and $r_{j,e} = 0$ otherwise.

We will return to this class of problems in Section VI, where we will compute suboptimal solutions and performance bounds for several instances of this problem.

V. Main Results

In this section we present a method for computing suboptimal solutions and performance bounds for the class of deterministic dynamic programs considered in this paper. First we will discuss a method for approximating the dynamic programming value function. For any given initial state x_0, this method computes an upper bound on the dynamic programming value function by solving a linear program. We will then show how to use this value function approximation as a surrogate for the dynamic programming value function, as in equation (3), leading to a mixed-integer linear programming method for computing suboptimal sequences. By comparing the reward achieved by the suboptimal sequence with the upper bound on achievable reward, we can determine how close the computed sequence is to optimal.

A. Implicit value function approximation

Recall that our goal is to determine a method for finding easily computable upper bounds on the optimal value of (1) for a given initial state x_0. Here we show that an upper bound on the optimal achievable value can be computed by solving a linear program with n continuous variables (where n is the dimension of the binary state vector x_t). This linear program is obtained by performing two relaxations: (i) a continuous relaxation of the binary variables x_t to obtain a semi-infinite linear program, and (ii) considering a restricted set of dual variables to relax this semi-infinite linear program to a linear program in n variables. This upper bound for a given x_0, which we will denote as $\widetilde{V}(x_0)$, will then be applied in an approximate dynamic programming procedure.

Lemma 2 in the appendix [6] shows that we can approximate $V(x_0)$ arbitrarily closely by considering finite horizon problems with sufficiently large horizon lengths. Specifically, let $V_N(x_0)$ denote the optimal value for the problem of horizon length N,

$$\begin{aligned} &\text{maximize: } \textstyle\sum_{t=0}^{N} \alpha^t r^T x_t \\ &\text{subject to: } A_1 x_{t+1} - A_2 x_t \le b \text{ for } t = 0, \ldots, N-1 \\ &\qquad\qquad x_t \in \{0,1\}^n \text{ for } t = 1, \ldots, N \end{aligned} \tag{7}$$

The derivation of $\widetilde{V}(x_0)$ will assume a given, arbitrary horizon length. After deriving a bound on the achievable value of (7), we will obtain our infinite horizon bound by letting $N \to \infty$.

Clearly, an upper bound on $V_N(x_0)$ can be obtained by solving a linear program with the same objective and constraints as (7), but $x_t \in [0,1]^n$ for all t. As N grows large, this upper bound on $V_N(x_0)$ approaches an upper bound on $V(x_0)$ (as we will soon show formally). However, the number of variables in the linear program relaxation of (7) grows correspondingly large as N increases. Since our goal is to obtain an easily computable upper bound on $V(x_0)$, we will introduce a second relaxation to bound the size of the problem we need to solve to obtain $\widetilde{V}(x_0)$.

The optimal value of the linear program relaxation of (7) can also be computed by solving its dual:

$$\begin{aligned} \text{minimize:} \quad & r^T x_0 + (A_2 x_0)^T \lambda_0 \\ & \quad + \sum_{t=0}^{N-1} b^T \lambda_t + \sum_{t=1}^{N} \mathbf{1}^T \mu_t \\ \text{subject to:} \quad & A_1^T \lambda_{t-1} - A_2^T \lambda_t + \mu_t \ge \alpha^t r \\ & \quad \text{for } t = 1, \ldots, N-1 \\ & A_1^T \lambda_{N-1} + \mu_N \ge \alpha^N r \\ & \lambda_t \ge 0 \text{ for } t = 0, \ldots, N-1 \\ & \mu_t \ge 0 \text{ for } t = 1, \ldots, N \end{aligned} \tag{8}$$

The derivation of this dual is straightforward and is provided in Lemma 3 in the appendix [6]. Furthermore, any feasible solution of (8) gives an upper bound on $V_N(x_0)$. So, one way to obtain a low-dimensional linear program for computing an upper bound on $V_N(x_0)$ is to search over a restricted set of feasible solutions of (8). In particular, we will restrict our search to solutions of

the form

$$\begin{aligned}\lambda_t &= \alpha^t\lambda \text{ for } t \in \{0,\ldots,N-1\}\\ \mu_t &= \alpha^t\mu \text{ for } t \in \{1,\ldots,N-1\}\\ \mu_N &= \alpha^N|r| + \alpha^{N-1}|A_1^T|\lambda\end{aligned}$$

This leads to the linear program in the variables $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^n$:

$$\begin{aligned}\text{minimize:}\quad & \left(r^T x_0 + \alpha^N \mathbf{1}^T|r|\right) + c_N^T\lambda + d_N^T\mu\\ \text{subject to:}\quad & (\alpha^{-1}A_1 - A_2)^T\lambda + \mu \geq \alpha r\\ & \lambda \geq 0\\ & \mu \geq 0\end{aligned} \tag{9}$$

where

$$\begin{aligned}c_N &= \left(\frac{1-\alpha^{N-1}}{1-\alpha}\right)b + A_2x_0 + \alpha^{N-1}|A_1|\mathbf{1}\\ d_N &= \left(\frac{\alpha-\alpha^{N-1}}{1-\alpha}\right)\mathbf{1}\end{aligned}$$

In the following theorem we show that the limit of the optimal value of (9) as $N \to \infty$ converges to an upper bound on $V(x_0)$. This is the main result of this section:

Theorem 1. *For given x_0, an upper bound on $V(x_0)$, which we call $\widetilde{V}(x_0)$, is obtained from the optimal value of the linear program*

$$\boxed{\begin{aligned}\text{maximize:}\quad & r^T x_0 + \alpha r^T z\\ \text{subject to:}\quad & (\alpha^{-1}A_1 - A_2)z \leq \tfrac{1}{1-\alpha}b + A_2x_0\\ & z \in \left[0, \tfrac{\alpha}{1-\alpha}\right]^n\end{aligned}} \tag{10}$$

Proof: Let $\widetilde{V}_N(x_0)$ denote the optimal value of the linear program (9), and let $\widetilde{V}(x_0)$ denote the optimal value of (10). We will first show that $\lim_{N\to\infty}\widetilde{V}_N(x_0)$ converges to $\widetilde{V}(x_0)$. Note that the dual of (10) is

$$\begin{aligned}\text{minimize:}\quad & r^T x_0 + (\tfrac{1}{1-\alpha}b + A_2x_0)^T\lambda + \tfrac{\alpha}{1-\alpha}\mathbf{1}^T\mu\\ \text{subject to:}\quad & (\alpha^{-1}A_1 - A_2)^T\lambda + \mu \geq \alpha r\\ & \lambda \geq 0\\ & \mu \geq 0\end{aligned} \tag{11}$$

Also, note that the sequence of vectors specifying the objective functions of (9) converges to the vector specifying the objective function of (11). Assuming that (7) is feasible for all x_0 and N, the dual relaxation (8) has a bounded solution for all N. Also, the feasible set of (11) is always nonempty ($\lambda = 0$ and $\mu = \alpha|r|$ is always feasible). Lemma 4 in the appendix [6] shows that under these conditions,

$$\lim_{N\to\infty}\widetilde{V}_N(x_0) = \widetilde{V}(x_0)$$

Also, by Lemma 2 in the appendix,

$$\lim_{N\to\infty}V_N(x_0) = V(x_0)$$

Finally, since $V_N(x_0) \leq \widetilde{V}_N(x_0)$ for all N we have

$$V(x_0) \leq \widetilde{V}(x_0)$$

■

In the next section we will show how this linear program for computing $\widetilde{V}(x_0)$ will provide a MILP-based method for computing a sequence of states.

B. Computing suboptimal solutions

In this section we present a method for computing suboptimal policies that use the approximate value function introduced in the previous section. This will result in a procedure for computing suboptimal policies that requires solving a mixed-integer linear program in each time period. While solving mixed-integer linear programs is generally considered to be a computationally difficult problem, we will address the advantages this method provides over enumerating the state space and computing exact value function. This approach will be further justified in the next section by showing the performance of this algorithm on two nontrivial problem instances.

Recall from the equation (2) that, if we had the dynamic programming value function V, we could compute an optimal sequence by solving a problem of the form (2) in each time period. If we instead have an approximation of V, say $\widetilde{V}$, then we can compute a suboptimal sequence by solving a problem of the form (3). According to Lemma 1, for any approximate value function $\widetilde{V}$ and any $\epsilon > 0$, there exists some sufficiently large N such that the solution to the problem (3) produces a sequence that achieves a reward within ϵ of $V(x_0)$. According to that Lemma, for any $\epsilon > 0$, the required N depends on the degree to which $\widetilde{V}$ approximates V.

Substituting our linear program for computing $\widetilde{V}$ into the equation (3), we obtain the following mixed-integer

linear program,

$$\begin{aligned}
\text{maximize:} \quad & r^T x_t + \textstyle\sum_{i=1}^{N} \alpha^i r^T x_{t+i} + \alpha^{N+1} r^T z \\
\text{subject to:} \quad & A_1 x_{i+1} - A_2 x_i \leq b \\
& \qquad \text{for } i = t, \ldots, t+N-1 \\
& -A_2 x_{t+N} + (\alpha^{-1} A_1 - A_2) z \leq \tfrac{1}{1-\alpha} b \\
& x_i \in \{0,1\}^n \text{ for } i = t, \ldots, t+N \\
& z \in \left[0, \tfrac{\alpha}{1-\alpha}\right]^n
\end{aligned}$$

In each time period t, x_t is known. Solving the MILP above provides the subsequent state x_{t+1}. The process is then repeated at time period $t+1$.

As long as the linear program (10) is a sharp upper bound on the infinite horizon value function, the performance bound in Lemma 1 guarantees that the sequence of states computed from repeated solution of this MILP will be nearly optimal. For example, in the job shop problem discussed in Section IV, the linear program (10) is similar to a multicommodity network flow relaxation. If this network flow relaxation provides a sufficiently tight approximation to the original discrete problem, then the MILP above will produce a nearly-optimal control sequence.

We will conclude this section by addressing the question of the computational requirements of solving mixed-integer linear programs in each time period. It might appear that we have simply replaced one hard problem (applying dynamic programming to a complex discrete-state system), with a series of different hard problems (solving a mixed-integer linear program at each time period). The applications that we are interested in have state variables with dimensions in the hundreds to low thousands. This leads to an astronomically large number of states, and application of dynamic programming requires specifying the value function at each state. On the other hand, applying the methods described in this section requires solving mixed-integer linear programs with a few hundred to a few thousand variables. Modern mixed-integer linear program solvers are capable of solving problems in this size range in a few seconds to a few minutes[1]. Moreover, the mixed-integer linear programs that are solved in consecutive time periods are very similar to one another. By using the solution from a previous period as a warm-start in the following period, we can significantly reduce the solution time at each period beyond $t = 0$.

[1]See http://plato.asu.edu/ftp/milpc.html for a recent performance analysis of the Gurobi and CPLEX solvers

	machine sequence	**reward**
Job 1:	$\mathbf{1}(3), \mathbf{4}(2), \mathbf{2}(1)$	1
Job 2:	$\mathbf{2}(5), \mathbf{1}(2), \mathbf{4}(1), \mathbf{3}(1)$	1

N	$U_N(x_0)$	$\widetilde{V}(x_0)$	**time (seconds)**
1	5.15	6.01	0.85
2	5.73	6.01	1.00
5	5.73	6.01	1.93
10	5.73	5.87	5.72
15	5.73	5.85	11.89
20	5.73	5.82	87.44

TABLE II
NUMERICAL EXAMPLE 1

VI. NUMERICAL EXAMPLES

In this section we apply the methods developed in this paper to two instances of the job shop problems described in Section IV. For the instance described in Section IV, we will provide a detailed summary of the running time and quality of solutions obtained for various values of the horizon length N. Following this, we will provide a detailed summary of the running time and quality of solutions for a more complex example. In both instances we evaluate schedules based on the total discounted reward associated with all completed jobs, where a discount factor of $\alpha = 0.95$ is used.

The first instance contains two job types, and is described in Table II (using the notation introduced in Section IV). The state of the system is described by a binary vector of dimension 20. Considering each realization of this vector to be a state, this yields a deterministic dynamic program with approximately 10^6 states. This is not a prohibitively large problem instance, and dynamic programming could probably be applied to this instance to compute an exact solution. The next instance we consider is significantly larger.

In Table II, we summarize the quality of the solutions and running times for various horizon lengths. $U_N(x_0)$ is used to denote the total discounted reward over an infinite time horizon. Since we use a discount factor of $\alpha = 0.95$, the the total discounted reward over 500 time periods is sufficient to approximate this quantity to within two decimal places. As before, $\widetilde{V}(x_0)$ denotes an upper bound on achievable infinite horizon reward. Also, we show the time in seconds required to compute 500 consecutive states $x_1, \ldots, x_{500}$ starting from a given x_0. The two numerical examples in this section were solved using GLPK in Octave on a MacBook Pro with a 2.4 GHz processor and 8 GB memory.

For a horizon of length $N = 1$, a solution is produced that is within 85% of optimal. The closeness to optimality for $N = 1$ may at first seem surprising, and is in fact

	machine sequence	reward
Job 1:	$\mathbf{1}(5), \mathbf{2}(3), \mathbf{1}(5), \mathbf{2}(2)$	1
Job 2:	$\mathbf{4}(5), \mathbf{1}(10), \mathbf{2}(5)$	1
Job 3:	$\mathbf{3}(5), \mathbf{2}(5), \mathbf{1}(6), \mathbf{2}(5)$	1
Job 4:	$\mathbf{4}(3), \mathbf{3}(4), \mathbf{2}(7), \mathbf{1}(5)$	1

N	$U_N(x_0)$	$\widetilde{V}(x_0)$	time (seconds)
1	0.95	2.02	9.56
2	1.47	1.95	20.87
5	1.47	1.76	72.79
10	1.52	1.69	240.89

TABLE III
NUMERICAL EXAMPLE 2

just a consequence of the fact that the value function approximation $\widetilde{V}(x_0)$ provides a close approximation to the true value function in this case. The optimality gap steadily decreases as we increase N, and for $N = 20$ a solution is produced that is guaranteed to be within 98% of optimal.

We will also present a second, more complex instance. This instance contains four job types, and each stage of processing requires between 2–10 time periods. As with the previous instance, this instance contains four machines (note that increasing the number of machines does not necessarily yield a more challenging problem instance). The state of the system can be described by a binary vector of dimension 86. Again considering each realization of this vector the state, this yields a deterministic dynamic program with approximately 7.7×10^{25} states. This is an extremely large state space, necessitating an approximate dynamic programming approach.

In Table III, we summarize the quality of solutions in running time as for the previous example. For this larger example, using the horizon of length $N = 1$ produces a solution that is guaranteed to be within 47% of optimal. A significant improvement is obtained by using a horizon of length $N = 2$, obtaining a solution guaranteed to be within 75% of optimal. The largest time horizon we consider, $N = 10$, yields a solution that is guaranteed to be within 89% percent of optimal.

VII. CONCLUSIONS

In this paper we considered an integer programming representation of infinite horizon deterministic dynamic programs. We provided a linear programming-based relaxation that can be used to compute an upper bound on the optimal dynamic programming value function at any state. By incorporating this value function approximation in an N-stage lookahead policy, we arrived at a mixed integer linear linear program that can be used to compute suboptimal policies and performance bounds. Through two numerical examples involving job shop scheduling over an infinite time horizon, we showed that this approach can quickly compute nearly optimal solutions.

REFERENCES

[1] E.J. Anderson and A.B. Philpott. *Infinite Programming*. Springer-Verlag, Berlin, 1985.

[2] A. Bemporad, F. Borrelli, and M. Morari. Model predictive control based on linear programming–the explicit solution. *IEEE Transactions on Automatic Control*, 47(12):1974–1985, 2002.

[3] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[4] D. Bertsekas. *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific, Cambridge, MA, 2000.

[5] W. Chen, D. Huang, A. Kulkarni, J. Unnikrishnan, Q. Zhu, P. Mehta, S. Meyn, and A. Wierman. Approximate dynamic programming using fluid and diffusion approximations with applications to power management. *To appear in the Proceedings of the 2009 IEEE Conference on Decision and Control*, 2009.

[6] R. Cogill and H. Hindi. `http://people.virginia.edu/~rlc9s/ACC11.pdf`. Online appendix to 'Computing Policies and Performance Bounds for Deterministic Dynamic Programs Using Mixed Integer Programming'.

[7] R. Cogill and H. Hindi. Optimal routing and scheduling in flexible manufacturing systems using integer programming. *Proceedings of the IEEE Conference on Decision and Control*, pages 4095–4102, 2007.

[8] M.A. Goberna and M.A. López. *Linear Semi-infinite Optimization*. Wiley, 1998.

[9] H. Hindi and W. Ruml. Network flow modeling for flexible manufacturing systems with re-entrant lines. In *IEEE Conf. on Dec. & Contr.*, December 2006.

[10] B. Lincoln and A. Rantzer. Relaxing dynamic programming. *IEEE Transactions on Automatic Control*, 51(8):1249–1260, 2006.

[11] P. Mehta and S. Meyn. Q-learning and Pontryagin's minimum principle. *To appear in the Proceedings of the 2009 IEEE Conference on Decision and Control*, 2009.

[12] M.L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. 2005, Springer.

[13] W.B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Wiley-Blackwell, 2007.

[14] C.V. Rao and J.B. Rawlings. Linear programming and model predictive control. *Journal of Process Control*, 10:283–289, 2000.

[15] W. Ruml, M.B. Do, and M.P.J. Fromherz. On-line planning and scheduling in a high-speed manufacturing domain. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*, 2005.

[16] P.J. Schweitzer and A. Seidmann. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.

[17] H. Topaloglu and W.B. Powell. Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. *INFORMS Journal on Computing*, 18(1):31, 2006.

[18] M. van den Briel and H. Hindi. A dynamic integer network flow model for reentrant flexible job shop scheduling. Technical report, Palo Alto Research Center (PARC), 2008.

[19] M.H.L. van den Briel, T. Vossen, and S. Kambhampati. Reviving integer programming approaches for ai planning: A branch-and-cut framework. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

[20] Y. Wang and S. Boyd. Performance bounds for linear stochastic control. *System and Control Letters*, 58(3):178–182, 2009.