# Approximate policy iteration for dynamic resource-constrained project scheduling

Mahshid Salemi Parizi, Yasin Gocgun, and Archis Ghate*

June 14, 2017

**Abstract**

We study non-preemptive scheduling problems where heterogeneous projects stochastically arrive over time. The projects include precedence-constrained tasks that require multiple resources. Incomplete projects are held in queues. When a queue is full, an arriving project must be rejected. The goal is to choose which tasks to start in each time-slot to maximize the infinite-horizon discounted expected profit. We provide a weakly coupled Markov decision process (MDP) formulation and apply a simulation-based approximate policy iteration method. Extensive numerical results are presented.

Keywords: Markov decision processes; approximate dynamic programming

## 1 Introduction and literature review

Mathematical studies on project scheduling date at least as far back as the Critical Path Method (CPM) and the Program Evaluation and Review Technique (PERT) [14]. Both CPM and PERT assume unlimited resource availability. Early work on resource-constrained project scheduling problems (RCPSPs) can perhaps be traced to the zero-one programming model in [16]. That paper studied a discrete-time problem and incorporated multiple projects with precedence-constrained tasks and multiple resources. Task durations were deterministic and new project arrivals were not allowed. The binary decision variables corresponded to whether or not a task is completed in a certain period. Such static-deterministic RCPSPs started receiving more attention in the 1980s. For instance, Blazewicz et al. [4] showed in 1983 that such RCPSPs are NP-hard. More recent surveys of static-deterministic RCPSPs are available in Herroelen et al. [10], and in Hartmann and Briskorn [9]. Hartmann and Briskorn stated that most of the literature available at the time focused on rather simplistic models and ignored two features that are important in practice: stochastic task durations and dynamic arrivals of new projects. One of the appropriate models for incorporating such stochastic and dynamic components is Markov decision processes (MDPs). We therefore review literature that uses MDPs to model project scheduling problems next.

Choi et al. [5] formulated an infinite-horizon, discrete-time MDP model for RCPSPs with stochastic task durations, uncertain task outcomes (success or failure), and uncertain costs. In their problem setting, only one resource was needed to perform one task. They focused on linear activity-on-node (that is, precedence constrained) networks and did not model dynamic arrivals of new projects. The state in their MDP included the status of each project, that is, which tasks

*Corresponding author: Industrial & Systems Engineering, BOX 352650, University of Washington, Seattle, Washington, 98195, USA; Phone: (206)-616-5968; Email: archis@uw.edu.

are complete and which are ongoing; information about whether or not the latest task in each project was a success was also stored in the state; in addition, the state included the number of time-slots for which a resource has been used for the currently ongoing task. Since their activity-on-node network was linear, at most one task in a project can be started in one time-period. The decisions in their MDP were therefore binary, representing whether or not to start a particular task in a particular project. The resulting MDP still suffered from the curse of dimensionality. A simulation-based approximate dynamic programming (ADP) algorithm was applied.

Choi et al. [6] extended the above MDP by allowing new project arrivals from a pre-determined group of projects. Consequently, their state included an additional variable to represent the realized arrival time of each new project. The decisions were identical to those in Choi et al. [5]. They implemented a variation of the $Q$-learning algorithm [17] on the resulting large-scale MDP.

Melchiors [11] formulated an infinite-horizon, continuous-time MDP model for scheduling dynamically arriving projects with stochastic task durations. Project arrivals followed a Poisson process. Their model was not restricted to linear activity-on-node networks. However, as in the above two papers by Choi et al., Melchiors also made the simplifying assumption that each task needed only one resource. The state in this MDP included information about waiting and ongoing tasks in each project. The decision-maker chose the tasks to work on in each period. A simulation-based ADP algorithm that employed value-function approximation was implemented.

We study infinite-horizon, discrete-time RCPSPs with dynamic arrivals of new heterogeneous projects. We allow for arbitrary arrival distributions and any (not necessarily linear) activity-on-node networks. One task may simultaneously require multiple resources. These features generalize the corresponding components of the problems studied in Choi et al. [5, 6] and in Melchiors [11]. We focus on deterministic task durations and provide an MDP formulation of such dynamic resource constrained project scheduling problems (DRCPSPs). It turns out that this MDP is weakly coupled [1]. That is, the immediate expected profits are additively separable over project-types and transition probabilities are multiplicatively separable. Decisions about distinct project-types are only linked by resource-availability constraints. Exact solution of this MDP is intractable owing to the curse of dimensionality. Standard approaches for approximate solution of weakly coupled MDPs include Lagrangian relaxation and approximate linear programming [1, 8, 12]. Unfortunately, owing to the complicated state-evolution process in our MDP, such mathematical programming-based methods are computationally difficult to implement. We therefore apply a simulation-based approximate policy iteration algorithm [3, 15] to this MDP. This method employs a value-function approximation whose parameters are tuned via simulation using least-squares fitting.

## 2 Problem statement

Consider a projection scheduling problem over time-stages $t = 1, 2, \ldots$ with the following notation.

1. $\mathcal{I} = \{1, 2, \ldots, I\}$ is the index set of project types.

2. Up to $D^i$ new projects of type $i \in \mathcal{I}$ may arrive during one time-period. Let $p^i(m)$ denote the probability that $m \in \{0, 1, \ldots, D^i\}$ new projects of type $i \in \mathcal{I}$ arrive during one time-period.

3. For each project of type $i \in \mathcal{I}$, $\mathcal{N}^i = \{1, \ldots, N^i\}$ denotes the finite set of tasks that need to be accomplished in order to complete the project. Tasks are performed in a non-preemptive manner; i.e., once started, a task cannot be interrupted until it is complete.

4. For each task $n \in \mathcal{N}^i$, $\mathcal{M}^i_n \subset \mathcal{N}^i$ denotes the set of tasks that must be completed before starting task $n$. This set is called the set of predecessors of task $n$ in project-type $i$.

5. Task durations are deterministic. Task $n \in \mathcal{N}^i$ takes $\Delta_n^i$ time-periods to complete.

6. $\mathcal{J} = \{1, \ldots, J\}$ is the set of resources. $B^j$ is the integer quantity of resource $j \in \mathcal{J}$ available in each time-period. Task $n \in \mathcal{N}^i$ requires an integer amount $b_n^{ij} \geq 0$ of resource $j \in \mathcal{J}$.

7. By the beginning of any time-period, a project that arrived earlier may be completed, may be incomplete or may be waiting inception. The projects that have not been completed, i.e., the ones that are incomplete or waiting inception form a "queue". $W^i < \infty$ denotes the queue capacity for incomplete and waiting projects of type $i$.

8. The following rewards and costs are obtained or incurred.
   - Projects of type $i \in \mathcal{I}$ that arrive when $W^i$ projects of that type are in the queue are rejected incurring a penalty cost $G^i$ at the end of the time-period.
   - A reward $R^i$ is received on completing a project of type $i \in \mathcal{I}$ at the end of a time-period.
   - A cost $c_n^i$ per time-period is charged at the end of that time period for performing task $n$ in project-type $i \in \mathcal{I}$.
   - An incomplete stalled project (no ongoing tasks) of type $i \in \mathcal{I}$ incurs a cost $Q^i$ per period. This cost is charged at the end of the time-period.
   - A holding cost $H^i$ per project of type $i \in \mathcal{I}$ is incurred in each time-period where such a project is waiting inception. This cost is incurred at the beginning of the time-period.

9. The goal is to maximize the total discounted expected profit (rewards minus costs) over an infinite-horizon where the per-period discount factor is $0 < \alpha < 1$.

An MDP model for this class of DRCPSPs is developed next.

## 3 A Markov decision process model

The MDP **states** are given by $X = (X^1, \ldots, X^I)$, where matrix $X^i$ stores information about all incomplete and waiting type-$i$ projects (i.e., projects in queue). The number of rows in $X^i$ equals the number of type-$i$ projects in queue and hence cannot exceed $W^i$. Let rows$(X^i)$ denote the set of rows in $X^i$. Recall that $\mathcal{N}^i = \{1, \ldots, N^i\}$ is the set of tasks in type-$i$ projects. Each row in $X^i$ is of length $N^i$. The entries $X_{lk}^i$ in the $l$th row and $k$th column of matrix $X^i$ are defined as

$$\begin{aligned} X_{lk}^i &= -1 \text{ if task } k \in \mathcal{N}^i \text{ in the } l\text{th type } i \text{ project in queue has not yet started;} \\ X_{lk}^i &= \Delta_k^i \text{ if task } k \in \mathcal{N}^i \text{ in the } l\text{th type-}i \text{ project in queue has been completed;} \\ 0 < X_{lk}^i &< \Delta_k^i \text{ if task } k \in \mathcal{N}^i \text{ in the } l\text{th type-}i \text{ project has started but not complete.} \end{aligned}$$

When $0 < X_{lk}^i < \Delta_k^i$, $X_{lk}^i$ denotes the number of time-periods since the inception of task $k$. Let $\mathcal{X}^i$ denote the set of all state matrices $X^i$ that are feasible with respect to precedence constraints for type-$i$ projects. Also let $\mathcal{X} = \mathcal{X}^1 \times \mathcal{X}^2 \times \ldots \times \mathcal{X}^I$. We define the set $\mathcal{C}_l^i(X^i) = \{k : 0 < X_{lk}^i < \Delta_k^i\}$ of tasks that have started but are incomplete. Owing to our non-preemptive setting, tasks in $\mathcal{C}_l^i(X^i)$ must be processed in the current period. The set of all feasible states is then given by

$$\bar{\mathcal{X}} = \left\{ X \in \mathcal{X} : \sum_{i=1}^{I} \sum_{l \in \text{rows}(X^i)} \sum_{k \in \mathcal{C}_l^i(X^i)} b_k^{ij} \leq B^j, j \in \mathcal{J} \right\}. \tag{1}$$

The vector $A$ of **action** matrices in our MDP is written as $A = (A^1, \ldots, A^I)$. Here, $A^i$ is a matrix of zeros and ones, equal in size to $X^i$, and indicates which tasks will be started next. $A^i_{lk} = 1$ implies that we choose to begin task $k$ in the $l$th type-$i$ project in queue; $A^i_{lk}$ is zero if we do not begin task $k$ in the $l$th type-$i$ project in queue. A feasible action must satisfy the logical restrictions

$$A^i_{lk} \;=\; 1 \text{ only if } X^i_{lk} = -1 \text{ and } X^i_{ln} = \Delta^i_n, \; \forall n \in \mathcal{M}^i_k. \tag{2}$$

This ensures that task $k$ can be started only if it had not begun or completed earlier and if all of its predecessors have been completed. The set of all such action matrices for state matrix $X^i$ is denoted by $\mathcal{U}^i(X^i)$. Also let $\mathcal{U}(X) = \mathcal{U}^1(X^1) \times \mathcal{U}^2(X^2) \times \ldots \times \mathcal{U}^I(X^I)$. Given the state-action pair $(X^i, A^i)$, the matrix $X^i + A^i$ provides valuable information. In particular, we define the set of ongoing tasks in the $l$th type-$i$ project as $\mathcal{V}^i_l(X^i, A^i) = \{k : 0 \le X^i_{lk} + A^i_{lk} < \Delta^i_k\}$. Note that $\mathcal{C}^i_l(X^i) \subseteq \mathcal{V}^i_l(X^i, A^i)$ for every $A^i \in \mathcal{U}^i(X^i)$. The amount of resource $j \in \mathcal{J}$ consumed by all ongoing tasks from type-$i$ projects equals $\sum\limits_{l \in \text{rows}(X^i)} \sum\limits_{k \in \mathcal{V}^i_l(X^i, A^i)} b^{ij}_k$ and is denoted by $B^{ij}\left(X^i, A^i\right)$. The set of all vectors of action matrices that are feasible in state $X \in \bar{\mathcal{X}}$ is defined as

$$\bar{\mathcal{U}}(X) = \left\{ (A^1, A^2, \ldots, A^I) \in \mathcal{U}(X) : \sum_{i=1}^{I} B^{ij}\left(X^i, A^i\right) \le B^j, \; j = 1, 2, \ldots, J \right\}. \tag{3}$$

Given the state-action pair $X^i, A^i$, **transitions** into the new state $X'^i$ are defined as follows. For the $l$th type-$i$ project in queue, the set of tasks waiting inception is defined as $\mathcal{E}^i_l(X^i, A^i) = \{k : X^i_{lk} + A^i_{lk} = -1\}$. The set of type-$i$ projects with no tasks waiting inception (i. e., all tasks either completed or ongoing) is defined as $\mathcal{F}^i(X^i, A^i) = \{l \in \text{rows}(X^i) : \mathcal{E}^i_l(X^i, A^i) = \emptyset\}$. A project $l \in \mathcal{F}^i(X^i, A^i)$ is completed by the end of the current time-period if all of its ongoing tasks are completed by then. The subset $\bar{\mathcal{F}}^i(X^i, A^i) \subseteq \mathcal{F}^i(X^i, A^i)$ denotes the type-$i$ projects that will be completed by the end of the current time-period. These projects do not appear in the next state. Recall that $m$ new type-$i$ projects arrive at the beginning of the current time-period with probability $p^i(m)$. Depending on how full the queue is at the time, none, some, or all of these will be added to the queue. The number of rows in the next state matrix $X'^i$ therefore equals $|\text{rows}(X'^i)| = |\text{rows}(X^i)| - |\bar{\mathcal{F}}^i(X^i, A^i)| + m - \left[|\text{rows}(X^i)| - |\bar{\mathcal{F}}^i(X^i, A^i)| + m - W^i\right]^+$, where the notation $|\cdot|$ is used throughout this paper for set cardinalities. For all tasks $k$ in all type-$i$ projects $l \in \text{rows}(X'^i)$ that newly joined the queue in this manner, we have, $X'^i_{lk} = -1$. For all ongoing tasks $k$ in each type-$i$ project $l \in \text{rows}(X^i)$, we have, $X'^i_{lk} = X^i_{lk} + 1$.

The **immediate expected profit** earned for type-$i$ projects on choosing a feasible action matrix $A^i$ in state matrix $X^i$ is given by

$$\begin{aligned}\phi^i(X^i, A^i) = \alpha R^i |\bar{\mathcal{F}}^i(X^i, A^i)| - \alpha \sum_{m=0}^{D^i} p^i(m) \left[|\text{rows}(X^i)| - |\bar{\mathcal{F}}^i(X^i, A^i)| + m - W^i\right]^+ G^i \\ - |\mathcal{W}^i(X^i, A^i)| H^i - \alpha |\mathcal{Z}^i(X^i, A^i)| Q^i - \alpha \sum_{l \in \text{rows}(X^i)} \sum_{k \in \mathcal{V}^i_l(X^i, A^i)} c^i_k.\end{aligned}$$

Here, the first term $\alpha R^i |\bar{\mathcal{F}}^i(X^i, A^i)|$ equals the discounted reward earned for completing projects in the set $\bar{\mathcal{F}}^i(X^i, A^i)$. The second term equals the discounted cost of rejecting projects that cannot fit into the queue. In the third term, $\mathcal{W}^i(X^i, A^i) = \{l \in \text{rows}(X^i) : X^i_{lk} + A^i_{lk} = -1 \; \forall k \in \mathcal{N}^i\}$ is the set of type-$i$ projects waiting inception. The third term thus accounts for the holding cost of projects that are waiting inception. In the fourth term, $\mathcal{Z}^i(X^i, A^i) = \{l : \mathcal{V}^i_l(X^i, A^i) = \emptyset, \; \mathcal{E}^i_l(X^i, A^i) \ne \emptyset, \; \mathcal{E}^i_l(X^i, A^i) \ne \mathcal{N}^i\}$ is the set of stalled type-$i$ projects. The fourth term thus equals the discounted cost of stalled projects. The last term is the discounted operating cost of all ongoing projects. The total immediate expected profit is denoted by $\phi(X, A) = \sum_{i=1}^{I} \phi^i(X^i, A^i)$.

Let $\sigma(X)$ be the maximum infinite-horizon discounted expected profit earned starting in state $X \in \bar{\mathcal{X}}$. Then, for all $X \in \bar{\mathcal{X}}$, Bellman's equations for our MDP are given by

$$\sigma(X) = \max_{A \in \bar{\mathcal{U}}(x)} \left\{ \phi(X, A) + \alpha \sum_{m^1=0}^{D^1} \cdots \sum_{m^I=0}^{D^I} \left( \prod_{i \in \mathcal{I}} p^i(m^i) \right) \sigma(X') \right\}. \tag{4}$$

Since the state and action sizes are exponential in $I$, in $W^i$ and in $N^i$ for $i \in \mathcal{I}$, exact solution of this MDP is computationally intractable.

The immediate expected profit $\phi(X, A)$ in this MDP is additively separable over $\mathcal{I}$, and the transition probabilities $\prod_{i \in \mathcal{I}} p^i(m^i)$ are multiplicatively separable over $\mathcal{I}$. State components $X^i$ and decisions $A^i$ for projects of distinct types $i$ from $\mathcal{I}$ are only connected through the linking state feasibility constraints (1) and resource constraints (3). Thus, this MDP is weakly coupled [1]. There are two main methods for approximate solution of weakly coupled MDPs: Lagrangian relaxation and linear programming. Unfortunately, owing to the large scale and highly combinatorial state transitions in our weakly coupled MDP, an efficient implementation of such mathematical programming-based methods or of their variations in [8, 12] does not seem possible. We instead resort to a simulation-based ADP method as described in the next section.

## 4 A simulation-based ADP method

There are four main ideas in the ADP method we describe here. First, we employ the notions of pre- and post-decision states as in [15] to partly handle the curse of dimensionality. Specifically, the state transitions from $X^i$ to $X'^i$ described above are partitioned into two pieces. The first piece corresponds to the state-transformation that results only from choosing action $A^i$. The second piece incorporates random project arrivals that occur after $A^i$ is selected. This considerably simplifies the Bellman's equations in (4). We then approximate the post-decision value function. Parameters of this value function are tuned via a simulation-based policy iteration approach that relies on least-squares fitting as in [2, 3, 15]. The resulting approximate value function is then substituted in the right hand side of the Bellman's equations to retrieve decisions in states that are visited in run-time. Mathematical details of this approach are described in the next three sections.

### 4.1 Pre- and post-decision states, and value function approximation

Given the pre-decision state $X^i$ and action $A^i$, the post-decision state $X^i(A^i)$ is defined as follows. The completed projects, that is, the projects in set $\bar{\mathcal{F}}^i(X^i, A^i)$, do not appear in the post-decision state. For all ongoing tasks $k$ in each type-$i$ project $l \in \text{rows}(X^i)$, we have, $X^i_{lk}(A^i) = X^i_{lk} + 1$. The effect of random project arrivals is then incorporated into this post-decision state to create the next pre-decision state $X'^i$. Let $X(A) = (X^1(A^1), \ldots, X^I(A^I))$. Then, based on this notion of a post-decision state, the Bellman's equations in (4) can be rewritten as

$$\sigma(X) = \max_{A \in \bar{\mathcal{U}}(X)} \left\{ \phi(X, A) + \alpha \sigma^A(X(A)) \right\}, \tag{5}$$

where $\sigma^A(X(A)) = E[\sigma(X') | X(A)]$. Here, the expectation is taken with respect to the probabilistic project arrival process. If the post-decision value function $\sigma^A(X(A))$ were known, the Bellman's equations in (5) only require the solution of a deterministic optimization problem. This is precisely the benefit of the concept of a post-decision state. As explained in [15], however, the post-decision value function needs to be approximated. We employ the parameterized approximation

$$\widetilde{\sigma}^A(X(A); \vec{r}) = r_0 + \sum_{i \in \mathcal{I}} \Big[ |\bar{\mathcal{F}}^i(X^i(A^i))| r_1^i + |\mathcal{W}^i(X^i(A^i))| r_2^i \Big]. \tag{6}$$

Here, $\bar{\mathcal{F}}^i(X^i(A^i))$ is the set of type-$i$ projects that will be completed at the end of the next period given that the post-decision state in this period is $X^i(A^i)$. Moreover, $\mathcal{W}^i(X^i(A^i))$ is the set of type-$i$ projects that will be waiting inception in the next pre-decision state given the post-decision state $X^i(A^i)$. Finally, $\vec{r} = (r_0; (r_1^1, r_2^1); \ldots; (r_1^I, r_2^I))$ is a vector of unknown parameters. In the next section, we apply an approximate policy iteration approach from [2, 3, 15] that uses least-squares fitting to adaptively tune these unknown parameters.

## 4.2 Implementation of least-squares approximate policy iteration

The policy iteration algorithm is initialized with an arbitrary parameter vector for the value-function approximation, which is updated in every iteration. Each iteration of the algorithm includes two main parts. The first part consists of creating $K$ sample paths each including $\tau$ time-steps. The $k$th sample path begins at an initial post-decision state. This $k$th sample path is generated by following a policy that is optimal with respect to the current value-function approximation. The discounted expected profit accumulated over $\tau$ steps in this sample path is calculated. In the second part of the algorithm, these $K$ profits then serve as input for a least-squares data fitting problem, where an optimal value-function parameter vector is derived. The new parameter vector is then calculated as a convex combination of the old and this optimal parameter vector before starting the next iteration. Details of this algorithm are listed below.

- **Step 0: Initialization**.
  - Step 0*a*. Initialize iteration $n = 1$; fix positive integers $\tau, K, N$; and fix a step-size $\gamma > 1$.
  - Step 0*b*. Initialize a parameter vector $\vec{r}^{\,1}$ arbitrarily.
- **Step 1: Policy evaluation through simulation**.
  - Step 1*a*. Set replication counter $k = 1$.
  - Step 1*b*. Warm-up: generate a post-decision $X_k^*$ and the next pre-decision $X_0$. Set $t = 0$.
  - Step 1*c*.
    * Find an optimal decision $A_t$ by solving

$$A_t = \underset{A \in \bar{\mathcal{U}}(X_t)}{\operatorname{argmax}} \Big\{ \phi(X_t, A) + \alpha \widetilde{\sigma}^A(X_t(A), \vec{r}^{\,n}) \Big\}. \tag{7}$$

    * Compute $\phi(X_t, A_t)$.
    * Obtain the next pre-decision state $X_{t+1}$ from $X_t$ and $A_t$.
    * If $t < \tau$, increment $t$ by one and go to Step 1c.
  - Step 1*d*. Compute discounted expected profit accumulated in periods $t = 0, 1, 2, \ldots, \tau$, by starting in state $X_k^*$ as $\psi(k) = \sum_{t=0}^{\tau} \alpha^t \phi(X_t, A_t)$.

– Step 1$e$. If $k < K$, increment $q$ and go to Step 1$b$.

- **Step 2: Value-function approximation update**

  – Step 2$a$. Find a parameter vector by solving the least-squares fitting problem $\vec{r}^{\,*} = \underset{\vec{r}}{\text{argmin}} \left\{ \sum_{k=1}^{K} \{\psi(k) - \widetilde{\sigma}(X_k^*, \vec{r}^{\,n})\}^2 \right\}$.

  – Step 2$b$. Let $\beta_n = \gamma/(\gamma + n - 1)$, and update $\vec{r}^{\,n+1} = (1 - \beta_n)\vec{r}^{\,n} + \beta_n \vec{r}^{\,*}$.

- **Step 3.** If $n < N$, increment $n$ by one and go to Step 1.

In numerical results, we implemented a myopic policy starting in an empty state to deliver the initial post-decision state $X_k^*$ in Step 1b of this algorithm. The step-size formula $\beta_n = \gamma/(\gamma+n-1)$ is from [15]. Finally, the deterministic optimization problem in (7) is identical in form to the online decision-retrieval problems. Our approach for solving these problems is described next.

## 4.3 Online retrieval of decisions

Let $\vec{r}^{\,*}$ be the final parameter vector after which the above algorithm is terminated. Suppose that we want to compute a decision vector that is optimal in a pre-decision state $X \in \bar{\mathcal{X}}$ with respect to the value function approximation (6). This is done by substituting $\widetilde{\phi}^A(X(A); \vec{r}^{\,*})$ from (6) in place of $\phi^A(X(A))$ in the right hand side of (5). This yields the deterministic optimization problem

$$\max_{A \in \bar{\mathcal{U}}(X)} \left\{ \phi(X, A) + \alpha \left( r_0^* + \sum_{i \in \mathcal{I}} \left[ |\bar{\mathcal{F}}^i(X^i(A^i))| r_1^{*i} + |\mathcal{W}^i(X^i(A^i))| r_2^{*i} \right] \right) \right\}. \tag{8}$$

The constant term $\alpha r_0^*$ from this problem can be ignored. By recalling the definitions of $\phi(X, A)$, $|\bar{\mathcal{F}}^i(X^i(A^i))|$ and $|\mathcal{W}^i(X^i(A^i))|$, it can be seen that they are not linear in components of $A$. By defining additional decision variables and associated logical constraints, we reformulate problem (8) so that it becomes linear. We first describe some new set notations.

Let $\mathcal{V}_l^i(X^i)$ denote the set of ongoing tasks in the $l$th type-$i$ project in state $X^i$. We use $\mathcal{E}_l^i(X^i)$ to denote the set of tasks in the $l$th type-$i$ project in state $X^i$, which have not been started yet. Similarly, let $\bar{\mathcal{E}}_l^i(X^i)$ denote the subset of tasks in $\mathcal{E}_l^i(X^i)$ that *could* be started because their predecessor tasks have been completed. Moreover, we use $\mathcal{W}^i(X^i)$ to denote the set of type-$i$ projects that are waiting inception in state $X^i$. Similarly, let $\mathcal{Z}^i(X^i)$ be the set of type-$i$ projects that are stalled in state $X^i$. Let $\mathcal{F}^i(X^i)$ denote the set of type-$i$ projects in which all tasks have been started in state $X^i$. Also let $\bar{\mathcal{F}}_1^i(X^i) \subseteq \mathcal{F}^i(X^i)$ be the type-$i$ projects that will be completed by the end of the current time-period because all incomplete tasks in these projects only need one time-period of additional work. Moreover, $\bar{\mathcal{F}}_2^i(X^i)$ is the set of type-$i$ projects in state $X^i$ that have the following properties: (i) predecessors of all tasks that have not been started yet are complete, and durations of all such tasks equal one time-period; and (ii) all ongoing tasks need only one time-period of additional work. If we chose to start all remaining tasks in a project in $\bar{\mathcal{F}}_2^i(X^i)$, then that project will be completed by the end of the current time-period. Similarly, let $\bar{\mathcal{F}}_3^i(X^i) \subseteq \mathcal{F}^i(X^i)$ be the type-$i$ projects that will be completed by the end of the next time-period because all incomplete tasks in these projects need at most two time-periods of additional work with at least one which needs two time-periods of additional work. Finally, let $\bar{\mathcal{F}}_4^i(X^i)$ be the set of type-$i$ in state $X^i$ with the following properties: (i) predecessors of all tasks that have not been started yet are complete, and durations of all such tasks equal either one or two time-periods; and (ii) all ongoing tasks need one or two time-periods of additional work; (iii) $\bar{\mathcal{F}}_2^i(X^i) \cap \bar{\mathcal{F}}_4^i(X^i) = \emptyset$.

We now describe new decision variables that can be recovered from a decision-matrix $A$, but defining them separately helps in linearizing the above problem. For every type-$i$ project $l$ in state $X^i$, we define binary decision variables $f_l^i$, $\bar{f}_l^i$, $w_l^i$, and $z_l^i$. The decision variable $f_l^i$ defines whether ($=1$) or not ($=0$) the $l$th type-$i$ project will be completed at the end of the *current* time-period. Similarly, $\bar{f}_l^i$ defines whether ($=1$) or not ($=0$) the $l$th type-$i$ project will be completed at the end of the *next* time-period. If the $l$th type-$i$ project is waiting inception, then the decision variable $w_l^i$ is 1 if we begin that project in this time-period; $w_l^i$ is 0 otherwise. If the $l$th type-$i$ project is stalled, decision variable $z_l^i$ is 1 if that project resumes in this time-period; $z_l^i$ is 0 otherwise. For every $m \in \{0, \ldots, D^i\}$, let $g_m^i = \max\{0, |\text{rows}(X^i)| - (|\bar{\mathcal{F}}_1^i(X^i)| + \sum_{l \in \bar{\mathcal{F}}_2^i(X^i)} f_l^i) + m - W^i\}$.

Problem (8) can then be rewritten as

$$\max_{A,f,\bar{f},w,z,g} \sum_{i=1}^{I} \Bigg( \alpha R^i(|\bar{\mathcal{F}}_1^i(X^i)| + \sum_{l \in \bar{\mathcal{F}}_2^i(X^i)} f_l^i) - \alpha \sum_{m=0}^{D^i} p^i(m) g_m^i G^i - (|\mathcal{W}^i(X^i)| - \sum_{l \in \mathcal{W}^i(X^i)} w_l^i) H^i$$

$$- (|\mathcal{Z}^i(X^i)| - \sum_{l \in \mathcal{Z}^i(X^i)} z_l^i) Q^i - \alpha \sum_{l \in \text{rows}(X^i)} ( \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i c_k^i + \sum_{k \in \mathcal{V}_l^i(X^i)} c_k^i) \Bigg) +$$

$$\alpha \sum_{i=1}^{I} \Bigg\{ r_1^{i*} \big( |\bar{\mathcal{F}}_3^i(X^i)| + \sum_{l \in \bar{\mathcal{F}}_4^i(X^i)} \bar{f}_l^i \big) + r_2^{i*} \big( |\mathcal{W}^i(X^i)| - \sum_{l \in \mathcal{W}^i(X^i)} w_l^i \big) \Bigg\}$$

subject to:

$$\sum_{i=1}^{I} ( \sum_{l \in \text{rows}(X^i)} ( \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i b_k^{ij} + \sum_{k \in \mathcal{V}_l^i(X^i)} b_k^{ij} ) \leq B^j, \; j \in \mathcal{J}, \tag{9}$$

$$f_l^i \leq A_{lk}^i, i \in \mathcal{I}, l \in \bar{\mathcal{F}}_2^i(X^i) \text{ and } k \in \bar{\mathcal{E}}_l^i(X^i), \tag{10}$$

$$f_l^i \geq \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i - |\bar{\mathcal{E}}_l^i(X^i)| + 1, i \in \mathcal{I}, l \in \bar{\mathcal{F}}_2^i(X^i), \tag{11}$$

$$\bar{f}_l^i \leq A_{lk}^i, i \in \mathcal{I}, l \in \bar{\mathcal{F}}_4^i(X^i) \text{ and } k \in \bar{\mathcal{E}}_l^i(X^i), \tag{12}$$

$$\bar{f}_l^i \geq \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i - |\bar{\mathcal{E}}_l^i(X^i)| + 1, i \in \mathcal{I}, l \in \bar{\mathcal{F}}_4^i(X^i), \tag{13}$$

$$w_l^i \leq \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i, i \in \mathcal{I} \text{ and } l \in \mathcal{W}^i(X^i), \tag{14}$$

$$w_l^i \geq \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i / |\bar{\mathcal{E}}_l^i(X^i)|, i \in \mathcal{I} \text{ and } l \in \mathcal{W}^i(X^i), \tag{15}$$

$$z_l^i \leq \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i, i \in \mathcal{I} \text{ and } l \in \mathcal{Z}^i(X^i), \tag{16}$$

$$z_l^i \geq \sum_{k \in \bar{\mathcal{E}}_l^i(X^i)} A_{lk}^i / |\bar{\mathcal{E}}_l^i(X^i)|, i \in \mathcal{I} \text{ and } l \in \mathcal{Z}^i(X^i), \tag{17}$$

$$g_m^i \geq |\text{rows}(X^i)| - (|\bar{\mathcal{F}}_1^i(X^i)| + \sum_{l \in \bar{\mathcal{F}}_2^i(X^i)} f_l^i) + m - W^i, \quad i \in \mathcal{I} \text{ and } m \in \{0, \ldots, D^i\}, \tag{18}$$

$$g_m^i \geq 0, \text{ integer}, \quad i \in \mathcal{I} \text{ and } m \in \{0, \ldots, D^i\}, \tag{19}$$

$$A_{lk}^i, \text{binary}, i \in \mathcal{I}, l \in \text{rows}(X^i) \text{ and } k \in \bar{\mathcal{E}}_l^i(X^i), \tag{20}$$

$$f_l^i, \text{binary}, i \in \mathcal{I}, l \in \bar{\mathcal{F}}_2^i(X^i), \tag{21}$$
$$\bar{f}_l^i, \text{binary}, i \in \mathcal{I}, l \in \bar{\mathcal{F}}_4^i(X^i), \tag{22}$$
$$w_l^i, \text{binary}, i \in \mathcal{I}, l \in \mathcal{W}^i(X^i), \tag{23}$$
$$z_l^i, \text{binary}, i \in \mathcal{I}, l \in \mathcal{Z}^i(X^i). \tag{24}$$

In this problem, (9) are resource constraints. Constraints (10)-(17) are logical restrictions that relate various decision variables as described next. Inequalities (10)-(11) are forcing constraints that relate $f_l^i$ and $A_{lk}^i$ for $l \in \bar{\mathcal{F}}_2^i(X^i)$ and $k \in \bar{\mathcal{E}}_l^i(X^i)$. Inequalities (12)-(13) are forcing constraints that relate $\bar{f}_l^i$ and $A_{lk}^i$ for $l \in \bar{\mathcal{F}}_4^i(X^i)$ and $k \in \bar{\mathcal{E}}_l^i(X^i)$. Inequalities (14)-(15) are forcing constraints that relate $w_l^i$ to $A_{lk}^i$ for $l \in \mathcal{W}^i(X^i)$. Finally, (16)-(17) are forcing constraints that relate $z_l^i$ to $A_{lk}^i$ for $l \in \mathcal{Z}^i(X^i)$. We use this linear integer program to recover decisions that are greedy with respect to the value function approximation parameterized by $\vec{r}^*$ in every visited pre-decision state. The same process is employed to solve problem (7) in Step 1c of the policy iteration algorithm.

## 5 Numerical results

All simulations were performed on a 3.1 GHz iMac desktop with 16 GB RAM and an Intel Core i7 chip running Mac OS X 10.9.3. All linear and linear integer programs were solved using CPLEX 12 from IBM via a MATLAB R2016a front-end.

We compared the performance of the simulation-based approximate policy iteration algorithm with a standard myopic policy. In any state $X \in \bar{\mathcal{X}}$, a myopic decision was obtained by solving the problem $\max\limits_{A \in \bar{\mathcal{U}}(X)} \phi(X, A)$, which was a linear integer program.

Our numerical experiments were performed on randomly generated problem instances as is common in the scheduling literature [8, 11, 12, 13]. This was done using a technique from [8, 12], which in turn, was an extension of a standard method to create multi-dimensional knapsack problems [7]. Since we have several new problem parameters that were not relevant in these earlier papers, we had to modify their problem generation approach as described next.

The discount factor was set to 0.85 as in [1, 8, 12]. The number of job types was fixed at $I = 5$, $I = 10$ and $I = 15$ in three different sets of problems. The number of resources was set to $J = 2$ in all instances. The number of tasks $N^i$ in type-$i$ projects was a randomly sampled integer between 5 and 25; we henceforth describe this process as $N^i \sim \mathrm{DU}[5, 25]$. The arrival probabilities were generated as in [8, 12]. Some of the other parameters were generated as follows: $W^i \sim \mathrm{DU}[1, 5]$; $R^i \sim \mathrm{DU}[5000, 10000]$; $G_i \sim \mathrm{DU}[1, 100]$; $H_i \sim \mathrm{DU}[1, 5]$; $Q_i \sim \mathrm{DU}[1, 5]$; $c_n^i \sim \mathrm{DU}[1, 3]$; $\Delta_n^i \sim \mathrm{DU}[1, 4]$. The activity-on-node network for each project-type was generated as a random lower triangular adjacency matrix with zeros on its diagonal. We also used $b_n^{ij} \sim \mathrm{DU}[1, 7]$. The resource availability $B^j$ was then set to $\lfloor I \times \rho \times \sum\limits_{i=1}^{I} \sum\limits_{n=1}^{N^i} b_n^{ij} \rfloor$ as in [7, 8, 12]. Here, $\rho$ is commonly called the tightness ratio. For each $I$, we used tightness ratios $\rho = 0.01, 0.02, 0.03, 0.04, 0.05, 0.07, 0.09, 0.1$. For each $I, \rho$ combination, 30 problem instances were generated randomly.

Step-size $\gamma$ was set to 10 as in Figure 6.3 from [15]. The replication parameter $Q$ was set to 50 as in [2, 3], and the simulation horizon was $\tau = 50$ within each replication. The number of iterations was $N = 100$. The warm-up duration to find $X_q^*$ was 50 periods.

Simulation-based policy iteration was compared with the myopic method via the average profit over 200 independent simulations with 50 periods each. Each of these simulations was started in an empty state. A simulation-based and a myopic decision were computed in each visited state. The next state was obtained by sampling a random number of new arrivals.

|  | $I = 5$ |  |  | $I = 10$ |  |  | $I = 15$ |  |  |
|---|---|---|---|---|---|---|---|---|---|
| $\rho$ | # sig. | # $S$ | % impr. | # sig. | # $S$ | % impr. | # sig. | # $S$ | % impr. |
| 0.01 | 22 | 8 | 2.6 | 20 | 13 | 4.1 | 19 | 15 | 9.2 |
| 0.02 | 21 | 12 | 6.9 | 18 | 13 | 3.12 | 20 | 16 | 5.9 |
| 0.03 | 17 | 11 | 7.9 | 20 | 18 | 1.5 | 21 | 17 | 1.3 |
| 0.04 | 7 | 6 | 7.8 | 25 | 22 | 2.4 | 23 | 20 | 6.7 |
| 0.05 | 11 | 8 | 6.23 | 21 | 19 | 9.5 | 17 | 11 | 3.8 |
| 0.07 | 9 | 5 | 2.3 | 25 | 24 | 2.6 | 23 | 19 | 3.1 |
| 0.09 | 13 | 6 | 0.09 | 26 | 24 | 5.4 | 16 | 14 | 1.9 |
| 0.1 | 12 | 8 | 4.4 | 25 | 20 | 7.37 | 19 | 17 | 3 |

Table 1: For each $I$, each row lists results for 30 randomly generated test instances. The column #sig. lists the number of instances (out of these 30) in which the difference between the two methods was statistically significant as per a paired $t$-test at threshold 0.05. The column #S lists the number of instances (out of these statistically significant ones) in which the simulation-based method generated a higher average profit than the myopic approach. The column % impr. reports the average percentage improvement achieved by the simulation-based method over the myopic in these instances.

Table 1 summarize our results for $I = 5, 10, 15$ with a variety of tightness ratios $\rho$. The table shows that out of the 720 problem instances listed, the difference between the two methods was statistically significant in 486, that is, in 67.5% of the problem instances. Of these 486 instances where the difference was significant, the simulation-based method outperformed the myopic approach in 377, that is, in 77.5% of the instances. In these instances, the simulation-based method outperformed the myopic method by about 4.5% on average. The tables also suggest that the simulation-based method performs better for $I = 10$ and $I = 15$ than for $I = 5$.

Our work here suggests that simulation-based policy iteration is a viable method for solving DRCPSPs. Future research could consider an extension where task-durations are stochastic.

## References

[1] D Adelman and A J Mersereau. Relaxations of weakly coupled stochastic dynamic programs. *Operations Research*, 56(3):712–727, 2008.

[2] D Astaraky. A simulation based approximate dynamic programming approach to multi-class, multi-resource surgical scheduling. Master's thesis, University of Ottawa, 2013.

[3] D Astaraky and J Patrick. A simulation based approximate dynamic programming approach to multi-class, multi-resource surgical scheduling. *European Journal of Operational Research*, 245(1):309–319, 2015.

[4] J Blazewicz, J K Lenstra, and A H G Rinooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[5] J Choi, M J Realff, and J H Lee. Dynamic programming in a heuristically confined state space: a stochastic resource-constrained project scheduling application. *Computers & Chemical Engineering*, 28(6):1039–1058, 2004.

[6] J Choi, M J Realff, and J H Lee. A Q-Learning-based method applied to stochastic resource constrained project scheduling with new project arrivals. *International Journal of Robust and Nonlinear Control*, 17(13):1214–1231, 2007.

[7] P C Chu and J E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(3):63–86, 1998.

[8] Y Gocgun and A Ghate. Lagrangian relaxation and constraint generation for allocation and advance scheduling. *Computers & Operations Research*, 39(10):2323–2336, 2012.

[9] S Hartmann and D Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[10] W Herroelen, B De Reyck, and E Demeulemeester. Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25(4):279–302, 1998.

[11] P Melchiors. *Dynamic and Stochastic Multi-Project Planning*, volume 673 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Switzerland, 2015.

[12] M Salemi Parizi and A Ghate. Multi-class, multi-resource advance scheduling with no-shows, cancellations, and overbooking. *Computers & Operations Research*, 67:90–101, 2016.

[13] J H Patterson. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30(7):854–867, 1984.

[14] M L Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Science & Business Media, New York, NY, USA, 2012.

[15] W B Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* John Wiley and Sons, Hoboken, New Jersey, USA, 2007.

[16] A A B Pritsker, L J Watters, and P M Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, 1968.

[17] C Watkins and P Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.