43rd IEEE Conference on Decision and Control
December 14-17, 2004
Atlantis, Paradise Island, Bahamas

Hierarchical Decision Making in Semiconductor Fabs using Multi-time Scale Markov Decision Processes

Jnana Ranjan Panigrahi and Shalabh Bhatnagar
Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India.
E-Mail: jnana@softjin.com, shalabh@csa.iisc.ernet.in

Abstract— **There are different timescales of decision making in semiconductor fabs. While decisions on buying/discarding of machines are made on the slower timescale, those that deal with capacity allocation and switchover are made on the faster timescale. We formulate this problem along the lines of a recently developed multi-time scale Markov decision process (MMDP) framework and present numerical experiments wherein we use TD(0) and Q-learning algorithms with linear approximation architecture, and show comparisons of these with the policy iteration algorithm. We show numerical experiments under two different scenarios. In the first, transition probabilities are computed and used in the algorithms. In the second, transitions are simulated without explicitly computing the transition probabilities. We observe that TD(0) requires less computation than Q-learning. Moreover algorithms that use simulated transitions require significantly less computation than their counterparts that compute transition probabilities.**

Index Terms— **Semiconductor fab, multi-time scale Markov decision process (MMDP), reinforcement learning, temporal difference (TD(0)) learning, Q-learning.**

I. Introduction

Different timescales of decision making arise naturally in semiconductor fabs. While decisions such as when to purchase additional machines or discard old ones are made less often or on a slower timescale, those that concern the day-to-day functioning of fab are made on a faster timescale. Decisions made on the slower timescale or higher-level affect the evolution of the faster timescale or lower-level process. Moreover, the lower-level performance metrics need to be taken into consideration for higher-level decisions. This problem has also been considered in [3], [4], [5], except that in these references, the above problem is formulated with all decisions viewed as being taken on a single timescale. For ease of computation, the numerical examples considered in [3], [4], [5] do not involve the higher-level processes at all, i.e., the capacities of various machines are assumed constant and decisions on buying new and/or discarding old machines are not considered.

Recently, in [6], a multi-time scale Markov decision process (MMDP) model has been developed and analyzed. Here the higher-level decisions are assumed to be taken only at certain multiples of lower-level decision epochs. Further, both higher-level and lower-level processes are assumed to be Markov decision processes (MDPs), however, the transition dynamics of the lower-level process depends also on the current higher-level state and action in addition to the same for lower-level. The transition dynamics of the higher-level process does not directly depend on the lower-level process. However, for selecting higher-level actions, an aggregate view of the lower-level costs incurred during a higher-level period is considered.

In this paper, we formulate the semiconductor fab decision making problem as an MMDP by describing both the higher and lower level transition dynamics in terms of state equations, constraints and the associated cost structure. Next we briefly describe the temporal difference (TD(0)) and Q-learning approaches for the MMDP setting. These are straightforward extensions of the same for Markov decision processes (MDPs) [1], [7]. We consider appropriate linear approximation architectures for both TD(0) and Q-learning algorithms. Proofs of convergence along with good error bounds for temporal difference learning with linear approximation architecture for the case of MDPs are available in [2], [8]. These can be shown to easily carry over for the MMDP setting as well. Next we present a numerical example wherein we consider a semiconductor fab modelled as an MMDP using the infinite horizon discounted cost formulation. We use both TD(0) and Q-learning algorithms to compute the optimal cost-policy pair. We consider two cases here. In the first, transition probabilities are computed and used in the corresponding algorithms. In the second case, transitions are simulated without explicitly computing the transition probabilities and this information is used in the algorithms. We show numerical comparisons of these algorithms with the policy iteration algorithm as well. The rest of the paper is organized as follows: In Section II, we describe in detail the problem formulation using the MMDP model. In Section III, we discuss the application of TD(0) with function approximation using a linear architecture. In Section IV, we discuss the Q-learning method (again) using linear approximation architecture. Finally, Section V presents the numerical results.

II. Model and Problem Formulation

Consider the problem of optimally controlling the entire life cycle dynamics of a semiconductor fab. An optimal decision rule should dictate for instance, when to buy new capacity (or discard old) and when to switch over capacity from one type of production and/or operation to another; see [3],[4],[5] for detailed discussions. In problems such

This work was supported in part by Grant number SR/S3/EE/43/2002-SERC-Engg from Department of Science and Technology, Government of India.

0-7803-8682-5/04/$20.00 ©2004 IEEE

as these, there is a clear hierarchy of decision making or difference in timescales that result from when (or at what times) individual decisions are made. For instance, decisions on buying/discarding machines are made much less often in comparison to those related to capacity switchover. In what follows, we first model this problem using the MMDP setting of [6] that is naturally applicable here.

A. The Model

We consider a discrete time model in which lower-level decisions are made at instants $0, 1, 2, \ldots$, while upper-level decisions are made once every T epochs, at instants $0, T, 2T, \ldots$, for some integer $T > 1$. The nth higher-level period (that we refer to as the nth decision period) thus comprises of instants nT, $nT + 1$, ..., $(n + 1)T - 1$. We call instants nT, $n \geq 0$, at which higher-level decisions are made as decision epochs. The tth lower-level period is simply referred to as the tth period. Before we proceed further, we first describe the notation used. Let 0 represent 'no operation'. Let P_n be the set of products manufactured during the nth decision period plus a 'no product' element (also denoted) 0, i.e., $0 \in P_n$. If a machine is idle, we say that it is manufacturing product 0, by performing operation 0. We call a *word* as any lexicographic ordering of operations with the first letter 0 in it such that there exists at least one machine in the fab that can perform all operations associated with it. Let $\mathcal{A}_n$ denote the set of all words in the nth decision period. We refer to the capacity associated with word w as type w capacity. In the following, the various quantities are indexed by terms of type (l, i) and w. These will be used to indicate the corresponding quantities associated with product l and operation i on machines of type w. We first describe the notation used.

Let $T_w(n)$ and $B_w(n)$ ($D_w(n)$) denote the total capacity and the amount of capacity bought (discarded), respectively, of type w, during the nth decision period. We also denote $C^a_w(x)$ ($C^b_w(x)$) as the cost of increasing (decreasing) type w capacity by purchasing (discarding) x units of new (old) capacity.

Let $X_{(l,i),w}(t)$ be the amount of available type w capacity allocated to product l, for operations of type $i \in w$ ($l \neq 0$, $i \neq 0$) in period t. Let K_w denote availability factor or the long-run average fraction of time that type w machines are available. Let $C_{(l,i),w}$ be number of finished wafers of product l after type i operations are performed on type w machines per unit time. It is a constant for given l, i and w. Also, $C_{(0,0),w} = 0 \; \forall \; w \in \mathcal{A}_n$. Let $F_{(l,i),w}$ be the total type i operations required for product l, on capacity of type w. Let $d_l(t)$ and $I_l(t)$ respectively denote the demand and inventory for product l in period t. Let $V_w^{(l,i),(m,j)}(t)$ be the amount of type w capacity switched over from product l and operations of type i to product m and operations of type j in period t with $i, j \in w$. The case $l = 0$, $i = 0$ signifies that the corresponding amount of capacity has been switched over from the available reserve and/or newly bought capacity of type w. Also, $m = 0$, $j = 0$ signifies that the above capacity has been switched over from product l and operations of type i to reserve or else has been permanently discarded. Let $C^e_w(x)$ ($C^e_w(x)$) be the switch over (operating) cost for x units of type w capacity. Also, let $C^d_l(x)$ be the inventory holding/backlogging cost for x units of product l.

B. State Dynamics for Higher-Level Process

The higher-level state (i_n) during the nth decision period is $i_n = (T_w(n),\ w \in \mathcal{A}_n)$. The higher-level action (λ_n) at time nT is $\lambda_n = (B_w(n),\ D_w(n),\ w \in \mathcal{A}_n)$. We denote the higher-level state and action spaces by I and Λ, respectively. We also denote by $\Lambda[i_n]$ the set of all feasible actions in (higher-level) state i_n. We have

$$T_w(n+1) = T_w(n) + B_w(n) - D_w(n),\ \forall \omega,\ \text{s.t.}$$
$$T_w(n), B_w(n) \geq 0,\ T_w(n) \geq D_w(n) \geq 0,$$

respectively. Note that the higher-level process here evolves in a deterministic manner as there is no source of randomness involved.

C. State Dynamics for Lower-Level Process

Given a higher-level state, there is a corresponding lower-level state space (in general a function of the higher-level state) that stays the same for T successive periods. We denote the state space at the lower-level by S_i when $i \in I$ is the corresponding higher-level state. A state at the lower-level denoted X_t in period t is the vector $X_t = (X_{(l,i),w}(t), I_l(t),\ (l,i) \in P_n \times w,\ w \in \mathcal{A}_n)$, where n is the corresponding decision period in which t is a period. A lower-level action denoted A_t in period t is defined by $A_t = (V_w^{(l,i),(m,j)}(t),\ l, m \in P_n,\ i, j \in w,\ w \in \mathcal{A}_n)$. Let U_i denote the set of all lower-level actions when the corresponding higher-level state is i. Also let $U_i(X)$ denote the set of all feasible lower-level actions when higher and lower level states are i and X, respectively. Then for t in the nth decision period, we have for $l \neq 0$, $i \neq 0$,

$$X_{(l,i),w}(t+1) = X_{(l,i),w}(t) + \sum_{\{(m,j) \in P_n \times w | (m,j) \neq (l,i)\}} (V_w^{(m,j),(l,i)}(t) - V_w^{(l,i),(m,j)}(t)),$$

$$I_l(t+1) = I_l(t) + \min_i \{ \sum_{\{w \in \mathcal{A}_n | F_{(l,i),w} > 0\}} \frac{1}{F_{(l,i),w}} C_{(l,i)w} X_{(l,i),w}(t) \} - d_l(t).$$

The middle term on the RHS of inventory equation above corresponds to throughput of product l. The constraints are as follows: $\forall\ w \in \mathcal{A}_n$,

$$\sum_{\{(m,j) \in P_n \times w | (m,j) \neq (l,i)\}} V_w^{(l,i),(m,j)}(t) \leq X_{(l,i),w}(t),$$
$$i \in w, l \in P_n, l \neq 0, i \neq 0,$$
$$\sum_{\{(m,j) \in P_n \times w | m \neq 0, j \neq 0\}} V_w^{(0,0),(m,j)}(t) \leq K_w T_w(n) -$$

$$\sum_{\{(m,j)\in P_n\times w|m\neq 0,j\neq 0\}} X_{(m,j),w}(t), w \in \mathcal{A}_n,$$

$$V_w^{(l,i),(m,j)}(t) \geq 0,\ (l,i),(m,j) \in P_n \times w.$$

Note that the only source of randomness in the lower-level process is the vector of demands $D_t = (d_l(t),\ l \in P_n)$. We assume that for t in the nth decision period, D_t may depend on slower and faster scale states and actions i_n, λ_n, X_t and A_t, respectively, according to some distribution $P(D_t \mid i_n, \lambda_n, X_t, A_t)$ (in general these probabilities can also be time inhomogeneous). Further, given i_n, λ_n, X_t and A_t; D_t is independent of $D_{t-1}, \ldots, D_0$.

D. Cost and Policies

If $i_n = i$, $\lambda_n = \lambda$, $X_t = X$ and $A_t = A$, then $X_{t+1} = Y$ according to probability $P^l(Y \mid X, A, i, \lambda)$. Note that this probability can be computed from the demand distribution $P(D_t \mid i, \lambda, X, A)$ above as described on pp.6 of [1]. The single stage cost for the lower-level MDP is given by

$$R^l(X_t, A_t, i_n, \lambda_n) = \sum_{w\in\mathcal{A}_n} \sum_{\{(l,i)\in P_n\times w\}} C_w^e(X_{(l,i),w}(t))$$

$$+ \sum_{l\in P_n} C_l^d(I_l(t)) + \sum_{w\in\mathcal{A}_n} \sum_{\{(l,i),(m,j)|l,m\in P_n; i,j\in w; (l,i)\neq(m,j)\}}$$

$$(C_w^c(V_w^{(l,i),(m,j)}(t))).$$

At the $(n+1)$st decision epoch, an upper level action λ' is chosen in state $i_{n+1} = i'$. This leads to a new lower level MDP starting with some initial state $Z \in X_{i'}$ at instant $t = (n+1)T$. We define a lower-level decision rule $d^l = \{\pi_n^l\}$, where $n = 0, 1, \ldots$, as a sequence of T-horizon nonstationary policies such that for all n, $\pi_n^l = \{\phi_{nT}, \ldots, \phi_{(n+1)T-1}\}$ is a set of mapping functions $\phi_k : S \times I \times \Lambda \longrightarrow A$ for all k in the nth decision period. Let D^l and D^U denote the sets of all posible decision rules for lower and higher-level MDPs, respectively. The single stage cost for the higher-level MDP is defined according to

$$R^u(X_{nT}, i_n, \lambda_n, \pi_n^l) = G(i_n, \lambda_n)$$

$$+E_{i_n,\lambda_n}^{X_{nT}}\{\sum_{t=nT}^{(n+1)T-1} R^l(X_t, \phi_t(X_t, i_n, \lambda_n), i_n, \lambda_n)\},$$

with X_{nT} as the initial lower-level state at time nT. Also, i_n and λ_n denote the higher-level state and action, respectively, during the nth decision period. Further,

$$G(i_n, \lambda_n) = \sum_{w\in\mathcal{A}_n} (C_w^a(B_w(n)) + C_w^b(D_w(n))),$$

thus the single-stage upper level cost is the sum of costs incurred in buying/discarding machines and all lower-level costs during a decision period. The objective here is to find optimal policies for slower and faster scale MDPs, d^u and d^l, respectively, in order to minimize the total expected infinite horizon α-discounted cost, where $\alpha \in (0,1)$. Thus

$$J^\star(X_0, i_0) = \min_{d^u\in D^u} \min_{d^l\in D^l} E_{d^u,d^l}^{X_0,i_0}\{\sum_{n=0}^{\infty} \alpha^n R^u(X_{nT}, i_n,$$

$$d^u(X_{nT}, i_n), \pi_n^l)\}$$

denotes the optimal cost. The Bellman equation for optimality can be written as in Theorem 1 of [6] as

$$J^\star(X,i) = \min_{\lambda\in\Lambda[i]} (\min_{\pi^l[i,\lambda]\in\Pi^l[i,\lambda]} \{R^u(X,i,\lambda,\pi^l[i,\lambda]) +$$

$$\alpha \sum_{j\in I} \sum_{Y\in S_j} P_{XY}^T(\pi^l[i,\lambda]) P^u(j \mid i,\lambda) J^\star(Y,j)\}), \quad (1)$$

where $\Pi^l[i,\lambda]$ is the set of all T-horizon lower-level policies corresponding to the higher-level state i and action λ, $\Lambda[i]$ is the set of admissible actions in state i, and $J^\star$ is the unique solution to the above equation for optimal $\lambda^\star$, and $\pi^{l,\star}[i,\lambda^\star]$, respectively. Here $P_{XY}^T(\pi^l[i,\lambda])$ corresponds to the T-step transition probability between lower-level states X and Y under lower-level policy $\pi^l[i,\lambda]$. Note that $P^u(j \mid i,\lambda)$ equals one if i,j,λ satisfy the higher-level state equation and constraint inequalities, else zero. We follow the procedure described in [6] for obtaining $P_{XY}^T(\pi^l[i,\lambda])$, $X \in S_i$, $Y \in S_j$. The key idea is to first replace $P_{XY}^T(\pi^l[i,\lambda])$ by a quantity $\delta(X,i,\lambda)[Y]$ that is computed according to

$$\delta(X,i,\lambda)[Y] = \sum_{Z\in S_i} P_{XZ}^{T-1}(\pi^l[i,\lambda]) \rho(Y \mid Z), \quad (2)$$

where $P_{XZ}^{T-1}(\pi^l[i,\lambda])$ is the $(T-1)$-step transition probability from X to Z corresponding to higher-level state i (itself) and action λ. Also $\rho(Y \mid Z)$ corresponds to probability of obtaining Y from Z with $Y \in S_j$. In our experiments, for transition from state $Z \in S_i$ at instant $(n+1)T-1$ to $Y \in S_j$ at instant $(n+1)T$, we use lower-level action u^* at $(n+1)T-1$ where $u^* = \arg\min_u R^l(Z,u,i,\lambda)$ s.t. $P_{ZY}^l(u^*) > 0$ for at least one $Y \in S_j$. Next probabilities $P_{ZY}^l(u^*)$ are normalized such that $\sum_{Y\in S_j} P_{ZY}^l(u^*) = 1$ by assigning zero probability to all $Y \notin S_j$. For obtaining lower-level optimal policies corresponding to a given higher-level state, we minimize only over the first term viz., $R^u(X,i,\lambda,\pi^l[i,\lambda])$ in the RHS of Bellman equation. Thus (under this approximation, see pp.981 of [6]), for given $\lambda \in \Lambda[i]$, $\pi^{l,\star}[i,\lambda] = \arg\min_{\pi^l[i,\lambda]\in\Pi^l[i,\lambda]} R^u(X,i,\lambda,\pi^l[i,\lambda])$, with control in the last period in a decision period selected as above. In many cases, T-step transition probabilities are hard to compute (particularly when T is large) but transitions can be easily simulated, a scenario that we also consider in this paper. In the latter case, we first simulate a lower-level state $Z \in S_i$ that can be reached from $X \in S_i$ in $T-1$ instants using the above policy. Next a state $Y \in S_j$ is obtained from $Z \in S_i$ in one step by simulating it using u^*. If $Y \notin S_j$, a new Y is generated in state $Z \in S_i$ until

the first Y in the set S_j is obtained. Further, all previous values of $Y \notin S_j$ are ignored.

III. Temporal-Difference Learning (TD(0)) with Function Approximation

A. Algorithm

Let $\tilde{i}_n$ denote (X_{nT}, i_n), the joint state process for lower and higher level MDPs observed at (higher-level) decision epochs. First select an infinite horizon arbitrary stationary policy $\pi^u = \{\mu, \mu, \ldots\}$ for selecting higher-level actions. We obtain states $\{\tilde{i}_n\}$ according to the above policy and transition mechanism as described before using either exact transition probabilities or by simulating transitions. Let $\alpha \in (0,1)$ be the discount factor. The cost-to-go function $J^\mu: S \rightarrow R$ associated with the given initial policy μ is defined by

$$J^\mu(\tilde{i}_0) \equiv E[\sum_{n=0}^{\infty} \alpha^n R^u(\tilde{i}_n, \mu(\tilde{i}_n), \pi^{l,*}[i_n, \mu(\tilde{i}_n)])].$$

Here we use a parameterized function $\tilde{J}^\mu: S \times I \times R^K \rightarrow R$ as an approximation to J^μ. We choose the parameter vector $r \in R^K$ so as to minimize the mean square error between $\tilde{J}^\mu(.,.,r)$ and $J^\mu(.,.)$. We update the parameter vector in the following manner for the joint process: Let r_n be the parameter vector at instant nT and d_n be the temporal difference corresponding to transition from $\tilde{i}_n$ to $\tilde{i}_{n+1}$, defined according to $d_n = R^u(\tilde{i}_n, \mu(\tilde{i}_n), \pi^{l,*}[i_n, \mu(\tilde{i}_n)]) + \alpha \tilde{J}^\mu(\tilde{i}_{n+1}, r_n) - \tilde{J}^\mu(\tilde{i}_n, r_n)$, for $n = 0, 1, \ldots$. Also, r_n is updated according to

$$r_{n+1} = r_n + \gamma_n d_n \nabla \tilde{J}^\mu(\tilde{i}_n, r_n),$$

where r_0 is some arbitrary vector, γ_n is the step-size and the gradient $\nabla \tilde{J}^\mu(\tilde{i}, r)$ corresponds to the feature vector for state $\tilde{i}$ because of the linear function approximator used as seen below. Note that $\tilde{J}^\mu(\tilde{i}, r)$ is of the following form with $r' = (r(1), \ldots, r(K))$.

$$\tilde{J}^\mu(\tilde{i}, r) = \sum_{k=1}^{K} r(k) \psi_k(\tilde{i}),$$

where $\psi_k(\tilde{i})$ is the kth component of the feature vector for state $\tilde{i}$. Let $\psi'(\tilde{i}) = (\psi_1(\tilde{i}), \ldots, \psi_K(\tilde{i}))$. Here for a vector x, x' denotes its transpose. Then $\tilde{J}^\mu(\tilde{i}, r)$ is given according to $\tilde{J}^\mu(\tilde{i}, r) = r'\psi(\tilde{i})$. Hence $\nabla \tilde{J}^\mu(\tilde{i}, r) = \psi(\tilde{i})$. We simulate states and simultaneously update the weight vector as above till the latter converges. After convergence of the weight vector, we use the approximate cost-to-go (with the converged weight vector) in the policy improvement step as follows:

$$\bar{\mu}(\tilde{i}) = \arg \min_{\lambda \in \Lambda[i]} (R^u(\tilde{i}, \lambda, \pi^{l,*}[i, \lambda]) + \alpha \sum_j \sum_Y \delta(\tilde{i}, \lambda)[Y] P^u(j \mid i, \lambda) \tilde{J}^\mu(\tilde{j}, r)),$$

where $\tilde{j} = (Y, j)$. After getting an improved policy, we use that policy again to obtain new states and update the weight vector. The above process is repeated until the policy converges.

B. Simulating Faster Timescale Path

It is difficult to find the transition probability matrix for the lower-level state process because of the large number of states. Hence as an alternative, we also consider the case where we simulate state transitions for lower-level states in the manner explained before. The policy improvement is then performed according to

$$\bar{\mu}(\tilde{i}) = \arg \min_{\lambda \in \Lambda[i]} (R^u(\tilde{i}, \lambda, \pi^{l,*}[i, \lambda]) + \alpha \tilde{J}^\mu(\tilde{j}, r))$$

with $\tilde{J}^\mu$ defined as before. The state $\tilde{j} = (Y, j)$ is obtained using simulation.

C. State Exploration

We observe in our experiments that some states are visited less frequently than others. The contribution of these states towards weight updates is thus less, implying that the weight vector will be biased by the states that are visited more often. Hence in our experiments, we also perform an ϵ-exploration of the less visited states. In other words, after certain time intervals, with a 'small' probability ϵ, states that are visited less are better explored and the chain is again continued. This is seen to improve performance.

IV. Q-learning with Function Approximation

For discounted problems, this algorithm converges to the optimal Q-factors (where Q-factors are defined as $Q(\tilde{i}, u) \triangleq R^u(\tilde{i}, u, \pi^{l,*}[i, u]) + \alpha \sum_j \sum_Y \delta(\tilde{i}, u)[Y] P^u(j \mid i, u) \min_v Q(\tilde{j}, v)$) provided all state-action pairs are visited infinitely many times. We update Q-factors $Q(\tilde{i}, u)$ according to

$$Q(\tilde{i}, u) := Q(\tilde{i}, u) + \gamma(R^u(\tilde{i}, u, \pi^{l,*}[i, u]) + \alpha \min_{v \in \Lambda[j]} Q(\tilde{j}, v) - Q(\tilde{i}, u)),$$

where $\tilde{j} = (Y, j)$ is the next state when action u and lower-level policy $\pi^{l,*}[i, u]$ are used in state $\tilde{i}$ and γ is the step-size parameter. We now consider a parameterized function approximation to the Q-factor by again considering a linear architecture. Let r be the weight vector as before. The approximate Q-factor is denoted $\tilde{Q}(\tilde{i}, u, r)$. Then the weight vector r is updated according to

$$r := r + \gamma \nabla \tilde{Q}(\tilde{i}, u, r)(R^u(\tilde{i}, u, \pi^{l,*}[i, u]) + \alpha \min_{v \in \Lambda[j]} \tilde{Q}(\tilde{j}, v, r) - \tilde{Q}(\tilde{i}, u, r)). \quad (3)$$

Let $\psi(\tilde{i}, u)$ be the feature vector for state-action pair $(\tilde{i}, u)$ such that $\psi'(\tilde{i}, u) = (\psi_k(\tilde{i}, u),\ k = 0, 1, \ldots, K)$. Then $\tilde{Q}(\tilde{i}, u, r) = r'\psi(\tilde{i}, u)$. Hence $\nabla \tilde{Q}(\tilde{i}, u, r) = \psi(\tilde{i}, u)$. We obtain a state trajectory $\{\tilde{i}_n\}$, with an arbitrarily chosen policy μ using either exact transition probabilities or simulated transitions. We initialize $r_0 = 0$. At time nT, suppose the weight is r_n, state is $\tilde{i}_n$ and higher-level action chosen

is u_n. The faster timescale component of state-vector $\tilde{i}$ is updated as explained earlier (using the δ-initialization function). Now suppose the next state is $\tilde{i}_{n+1}$ and we select an admissible action u_{n+1}. Then the weight vector is updated as follows:

$$r_{n+1} = r_n + \gamma_n \psi(\tilde{i}_n, u_n)(R^u(\tilde{i}_n, u_n, \pi^{l,*}[i_n, u_n]) + \alpha \tilde{Q}(\tilde{i}_{n+1}, u_{n+1}, r_n) - \tilde{Q}(\tilde{i}_n, u_n, r_n)). \quad (4)$$

The above is similar to the TD(0) algorithm with linear architecture, hence it converges if all state-action pairs are visited infinitely many times and an appropriate step-size is chosen. The action u_{n+1} is chosen with a certain probability $1 - \epsilon$ to be the one that minimizes $\tilde{Q}(\tilde{i}_{n+1}, u, r_n)$ over $u \in \Lambda[i_{n+1}]$ and is randomly chosen with the remainder probability of ϵ (see, for instance, pp.338-339 of [2]) for some small ϵ. The step size γ_n is set according to $\frac{c}{z_n(\tilde{i},u)+b}$, where $z_n(\tilde{i}, u)$ is the number of times the state-action pair $(\tilde{i}, u)$ is visited during $\{0, 1, \ldots, n\}$ with $c, b > 0$. The simulation is run until the weight vector converges to say r^*. Next we obtain optimal policy $\tilde{\mu}$ according to

$$\tilde{\mu}(\tilde{i}) = \arg \min_{u \in \Lambda[i]} \tilde{Q}(\tilde{i}, u, r^*).$$

V. Numerical Experiments

Consider a semi-conductor fab manufacturing products A and B with each requiring operations 'litho' and 'etch', respectively. We assume machines are either litho or etch and are of flexible type i.e., each one of these can perform the corresponding operation (viz., litho or etch) on both products. A similar model (as for the lower-level states here) has been considered in [3], [4], [5]. However, unlike here, the higher-level dynamics has not been considered in the above references but is assumed frozen. The higher-level state has the form $\{(T_{litho}, T_{etch})\}$, which is a vector of total capacities associated with 'litho' and 'etch' operations respectively. We assume that the components of the higher-level state vector take values from the following set:

$$T_{litho}, T_{etch} \in \{2, 3\}.$$

Thus the total number of possible higher-level states is 4. The higher-level action vector has the form $(B_{litho}, B_{etch}, D_{litho}, D_{etch})$, where we also assume that

$$B_{litho}, B_{etch}, D_{litho}, D_{etch} \in \{0, 1\}.$$

Thus at any given instant, only one machine of a given type can be bought or discarded. We also assume for simplicity that we do not buy and discard the same type of machine in a period. We have the following higher-level actions: $\lambda_1 = (1,1)$, $\lambda_2 = (1,0)$, $\lambda_3 = (1,-1)$, $\lambda_4 = (0,1)$, $\lambda_5 = (0,0)$, $\lambda_6 = (0,-1)$, $\lambda_7 = (-1,1)$, $\lambda_8 = (-1,0)$ and $\lambda_9 = (-1,-1)$, respectively. The first component above stands for litho capacity and the second for etch. The value $+1$ (resp. -1) indicates that one unit of the corresponding capacity is bought (resp. discarded). At the lower-level, we assume that no capacity is sent to reserve during the entire decision horizon and availability factor is 100%. Thus we assume that machines neither fail nor are sent for maintenance. The lower-level state has four components – litho capacity for product A, etch capacity for product A, inventory of A and inventory of B, respectively. Note that the litho and etch capacities for product B are thus automatically set. We assume that inventory for products A and B are constrained according to $I_A \in \{-1, 0, +1\}$ and $I_B \in \{-0.5, 0, +0.5\}$, respectively. We use both TD(0) as well as Q-learning algorithms for both cases where the transition probabilities are computed and used (as explained earlier) in the algorithm and where these are not computed but transitions are directly simulated, and also compare the results obtained using these with the policy iteration algorithm. It is easy to see that we have a total of 81, 108, 108 and 144 lower-level states corresponding to the upper-level states $(2,2)$, $(2,3)$, $(3,2)$ and $(3,3)$, respectively. Thus in the setting considered here, we find the optimal policy and value function for a total of 441 possible states and 9 actions. We assume that the demands for products A and B are independent random variables that are both distributed according to the Bernoulli distribution (though with different parameters) (see also [3], [4], [5]).

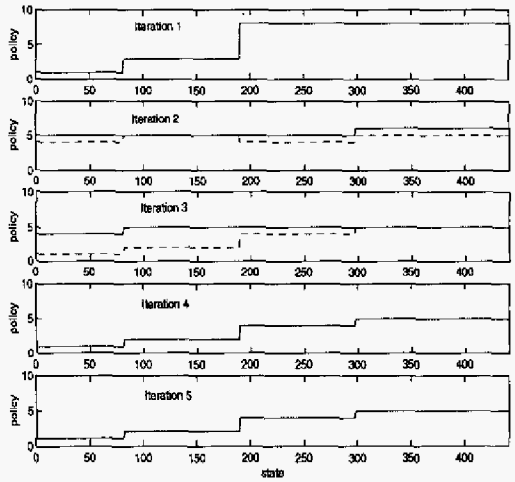

Fig. 1. Policy Updates for TD(0) with Transition Probabilities and Simulated Transitions Respectively

We used the following values for the various quantities in the experiments. The value of T is chosen to be 100. The cost for buying (discarding) unit capacity equals 100 (40). We select $Pr(D_A = 1) = 0.4 = 1 - Pr(D_A = 2)$. Also, $Pr(D_B = 0.5) = 0.7 = 1 - Pr(D_B = 1)$. The unit inventory cost for product A (B) equals 2 (1). The unit backlog cost for product A (B) equals 10 (5). The unit operating cost on litho (etch) machine for both products A and B is 0.2 (0.1). Also, the unit switch over cost on both litho and etch machines is 3. We assume the cost functions $C_w^a(\cdot)$, $C_w^b(\cdot)$, $C_w^c(\cdot)$, $C_l^d(\cdot)$ and $C_w^e(\cdot)$ to be linear in their arguments. We selected a total of thirteen features for algorithm TD(0). The first feature chosen is 1

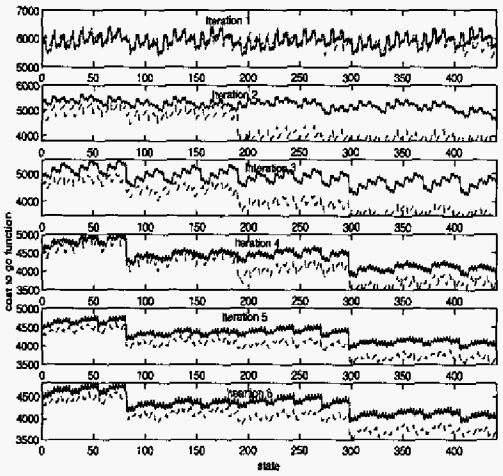

Fig. 2. Cost-to-go Function Updates for TD(0) with Transition Probabilities and Simulated Transitions Respectively

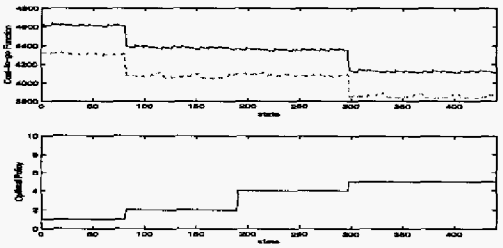

Fig. 3. Optimal Costs-to-go and Policies with Q-learning with Transition Probabilities and Simulated Transitions Respectively

while the rest are state values and their squares. We thus use a quadratic polynomial approximation of the cost-to-go function. This is a reasonable choice since we know that any continuous function on a compact set can be approximated arbitrarily closely by a polynomial. The policy and cost-to-go iterations using TD(0) are shown in Figs.1 and 2, respectively. Plots with solid lines (in all figures here) indicate the case when transition probabilities are known and the ones with broken lines indicate simulated transitions. Fig.4 shows policy and cost-to-go iterations using the policy iteration algorithm. Observe that the optimal policy obtained using all algorithms is exactly the same. Moreover, the cost-to-go functions using all algorithms have similar structures except that when compared with the policy iteration algorithm, there seems to be greater variability in costs obtained using TD(0) in both cases.

For the Q-learning algorithm, we select features for each state-action pair as follows: For states, the first feature is 1 followed by all state values of the joint process and their squares. The action features are only the values of the action vector. Thus the total number of features for each state-action pair is 15. The feature vector matrix is of full rank and the feature vectors for all state and action pairs are independent of each other. The optimal cost-to-go and policy

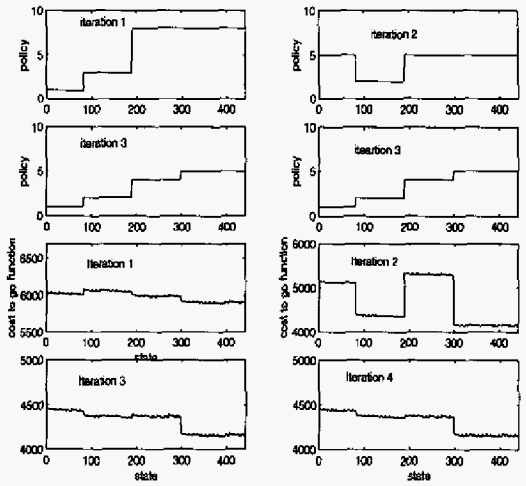

Fig. 4. Policy and Cost-to-go Updates using Exact policy iteration

obtained using Q-learning for both cases is shown in Fig.3. The optimal policy is again the same as before. Note also that the cost-to-go function here has less variability as is the case with policy iteration. The simulation based versions of both TD(0) and Q-learning algorithms show lower values for the optimal cost-to-go function than their counterparts that require knowledge of transition probabilities. Moreover the simulation based versions are computationally more efficient than their counterparts. On an IBM workstation with Linux operating system using the C programming language, simulation based TD(0) and Q-learning algorithms require about 15 and 30 minutes, respectively, for convergence, while the transition probability variants of these require about 3-4 times more computational time as compared to their respective counterparts that use simulated transitions.

REFERENCES

[1] D.P. Bertsekas, *Dynamic Programming and Optimal Control*, Vol. I, Athena Scientific, 1995.

[2] D.P. Bertsekas and J.N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, 1996.

[3] S. Bhatnagar, E.Fernandez-Gaucherand, M.C. Fu, S.I. Marcus and Y. He, "A Markov Decision Process Model for Capacity Expansion and Allocation", *Proceedings of the 38th IEEE Conference on Decision and Control*, pp.1156-1161, Phoenix, Arizona, 1999.

[4] S. Bhatnagar, M.C. Fu, S.I. Marcus, and Y. He, "Markov Decision Processes for Semiconductor Fab-Level Decision Making", *Proceedings of the IFAC 14th Triennial World Congress*, pp.145-150, Beijing, China, 1999.

[5] Y.He, S. Bhatnagar, M.C. Fu and S.I. Marcus, " Approximate Policy Iteration for Semiconductor Fab-Level Decision Making - a case study", Technical Report, Institute for Systems Research, University of Maryland, URL: http://www.isr.umd.edu/TechReports/ISR/2000/TR_2000-49/, 2000.

[6] H.S.Chang, P.Fard, S.I. Marcus, and M. Shayman, "Multi-time Scale Markov Decision Processes", *IEEE Transactions on Automatic Control*, 48(6):976-987, 2003.

[7] M.L.Puterman, *Markov Decision Processes*, John Wiley and Sons, New York, 1994.

[8] J.N. Tsitsiklis and B.V.Roy, "An analysis of temporal difference learning with function approximation", *IEEE Transactions on Automatic Control*, 42(5):674-690, 1997.