

Simulation-Based Approach for Semiconductor Fab-Level Decision Making - Implementation Issues

Ying He yhe@isr.umd.edu
Michael C. Fu mfu@isr.umd.edu
Steven I. Marcus marcus@isr.umd.edu
Institute for Systems Research
University of Maryland
College Park, MD 20742
<http://www.isr.umd.edu/IPDPM/>

Abstract

In this paper, we discuss implementation issues of applying a simulation-based approach to a semiconductor fab-level decision making problem. The fab-level decision making problem is formulated as a Markov Decision Process (MDP). We intend to use a simulation-based approach since it can break the “curse of dimensionality” and the “curse of modeling” for an MDP with large state and control spaces. We focus on how to parameterize the state space and the control space.

Keywords: Simulation-Based Approach, Markov Decision Processes, Semiconductor Fab-Level Decision Making, Cost Model, Demand Model.

1 Introduction

Problems of sequential decision making under uncertainty are common in manufacturing, computer and communication systems. Many such systems are very large and complicated. Consider a semiconductor fab capable of producing various wafers. The manufacturing process performed on each wafer contains a few hundred process steps and involves many types of equipment. Each piece of equipment can be used for various steps, and any given step can be executed on various pieces of equipment, perhaps at different rates. The complication is exacerbated by uncertainties such as frequent successive advances in technology and continual changes in demands for products [1] [2]. Usually, the decision making of a semiconductor fab is carried out according to a general hierarchical framework based on a temporal and/or physical decomposition of the system. In our IPDPM (Integrating Product Dynamics and Process Models) project on planning and scheduling of semiconductor manufacturing fabs, we attempt to deal with decision making at the fab level, and issues that must be addressed at this level include, for example, when to add additional capacity and when to convert from one type of production to another [3].

For fab-level decision making, our approach is to formulate it as a Markov Decision Process (MDP) [4] by defining appropriate states, actions, transition probabilities, time horizon, and cost criterion. The methodology for solving MDPs is dynamic programming. However, many decision making problems such as our fab-level decision making problem involve a large state space and/or a large control space (“curse of dimensionality”), which leads to difficulties on computation and storage of cost functions. In addition, dynamic programming requires an explicit model for the cost structure and the transition probabilities of the system, but such model is not accessible in some systems (“curse of modeling”). This happens to the fab-level decision making problem too.

In order to break the “curse of dimensionality” and the “curse of modeling”, we will discuss some simulation-based algorithms based on the following ideas: 1) simulation for the above mentioned systems is possible, where the task to evaluate the costs (cost-to-go, or average cost and differential costs) is estimated from transitions on simulated sample paths; 2) compact representations can be used for cost-to-go functions; 3) policies can be parameterized.

Then, we will propose ways to implement simulation-based algorithms on the fab-level decision making problem. Some key issues on implementation are how to build an appropriate compact representation of the cost-to-go function and how to parameterize the policy.

2 Markov Decision Processes

A Markov Decision Process is a framework containing states, actions, costs, probabilities and the decision horizon for the problem of optimizing a stochastic discrete-time dynamic system. The dynamic system equation is

$$x_{t+1} = f_t(x_t, u_t, w_t), \quad t = 0, 1, \dots, T - 1, \quad (1)$$

where t indexes a time epoch; x_t is the state of the system; u_t is the action to be chosen at time t ; w_t is a random disturbance which is characterized by a conditional probability distribution $P(\cdot | x_t, u_t)$; and T is the decision horizon. We denote the set of possible system states by S and the set of allowable actions in state $i \in S$ by $U(i)$. We assume S , $U(i)$, and $P(\cdot | x_t, u_t)$ do not vary

with t . We further assume that the sets S and $U(i)$ are finite sets, where S consists of n states denoted by $0, 1, \dots, n-1$.

If, at some time t , the system is in state $x_t = i$ and action $u_t = u$ is applied, we incur a stage cost $g(x_t, u_t) = g(i, u)$, and the system moves to state $x_{t+1} = j$ with probability $p_{ij}(u) = P(x_{t+1} = j \mid x_t = i, u_t = u)$. $p_{ij}(u)$ may be given a priori or may be calculated from the system equation and the known probability distribution of the random disturbance. $g(i, u)$ is assumed bounded.

Consider the **stochastic shortest path problem**, in which it is assumed that there is a special cost-free termination state n in the system and the system remains there at no further cost once it reaches that state. The objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \dots\}$ with $\mu_t : S \rightarrow U, \mu_t(i) \in U(i)$ for i and t , the total expected cost,

$$J_\pi(i) = \lim_{T \rightarrow \infty} E \left\{ \sum_{t=0}^{T-1} g(x_t, \mu_t(x_t)) \mid x_0 = i \right\}. \quad (2)$$

A stationary policy is an admissible policy of the form $\pi = \{\mu, \mu, \dots\}$; we denote it by μ_∞ .

The methodology for solving MDPs is dynamic programming, based on Bellman's "Principle of Optimality" [4]. One algorithm for solving MDPs is policy iteration. Policy iteration consists of a sequence of policy evaluation and policy improvement at each iteration. At each iteration step k , a stationary policy $\mu_\infty^k = \{\mu^k, \mu^k, \dots\}$ is given.

1. **Policy evaluation:** obtain the corresponding cost-to-go $J_{\mu^k}(i)$ by solving a linear equation.
2. **Policy improvement:** find a stationary policy $J_{\mu^{k+1}}$, where for all i , $\mu^{k+1}(i)$ is such that

$$g(i, \mu^{k+1}(i)) + \sum_{j=0}^{n-1} p_{ij}(\mu^{k+1}(i)) J_{\mu^k}(j) = \min_{u \in U(i)} [g(i, u) + \sum_{j=0}^{n-1} p_{ij}(u) J_{\mu^k}(j)]. \quad (3)$$

If $J_{\mu^{k+1}} = J_{\mu^k}$ for all i , the algorithm terminates; otherwise, the process is repeated with μ^{k+1} replacing μ^k .

Under certain assumptions, the policy iteration algorithm terminates in a finite number of iterations with a stationary optimal policy.

Another method of solving the optimality equations is value iteration. It is done by using the recursion

$$J_{k+1}(i) = \min_{u \in U(i)} [g(i, u) + \sum_{j=0}^{n-1} p_{ij}(u) J_k(j)], \quad i = 0, \dots, n-1. \quad (4)$$

given any initial conditions $J_0(0), \dots, J_0(n-1)$.

Consider **the finite horizon problem**, where T is finite. The objective is to minimize over all policies $\pi = \{\mu_0, \mu_1, \dots, \mu_{T-1}\}$ with decision rules $\mu_t : S \rightarrow U, \mu_t(i) \in U(i)$ for i and t , the total expected cost,

$$J_\pi(i) = E \left\{ G(x_T) + \sum_{t=0}^{T-1} g(x_t, \mu_t(x_t)) \mid x_0 = i \right\}. \quad (5)$$

The optimality equations are given by:

$$\begin{aligned} J_T^*(i) &= G(i); & i &= 0, \dots, n-1 \\ J_t^*(i) &= \min_{u \in U(i)} \sum_{j=0}^{n-1} p_{ij}(u) (g(i, u, j) + J_{t+1}^*(j)). \end{aligned} \quad (6)$$

Note that the finite horizon problem can be converted into a stochastic shortest path problem by viewing time as an extra component of the state. In the reformulation, transitions occur from state-time pairs $[i, t]$ to state-time pairs $[j, t + 1]$ according to the transition probabilities $p_{ij}(u)$ of the finite horizon problem; the termination state corresponds to the end of the horizon; it is reached in a single transition from any state-time pair of the form $[j, T]$ at a terminal cost $G(j)$ [5]. The reformulation is as follows:

$$\begin{aligned} J^*([i, T]) &= G(i); & i = 0, \dots, n - 1 \\ J^*([i, t]) &= \min_{u \in U([i, t])} \sum_{j=0}^{n-1} p_{ij}(u)(g(i, u, j) + J^*([j, t + 1])). \end{aligned} \quad (7)$$

So potentially, policy iteration or value iteration algorithms for the stochastic shortest path problem can be applied to a finite horizon problem.

3 Simulation-Based Algorithms

There are many simulation-based algorithms in the literature. Some of them are derived from policy iteration or value iteration, and they are called simulation-based policy iteration or simulation-based value iteration algorithms.

Simulation-based policy iteration algorithms have the same structure as exact policy iteration except for two differences [5]:

- Given the current policy μ , the corresponding cost-to-go function J_μ is not computed exactly. Instead, an approximate cost-to-go function $\tilde{J}_\mu(i, r)$ is computed, where r is a vector of tunable parameters.
- Once approximate policy evaluation is completed and $\tilde{J}_\mu(i, r)$ is available, we generate a new policy $\tilde{\mu}$ which is greedy with respect to \tilde{J}_μ . The greedy policy can be calculated exactly, or approximated.

There are several ways to estimate $\tilde{J}_\mu(i, r)$, such as least squares, Kalman filtering and temporal-difference learning etc. (see [5] chapter 3).

A simulation-based version of value iteration, often referred to as Q-learning, updates directly estimates of the Q-factors associated with an optimal policy. The optimal Q-factor is defined as $Q^*(i, u) = E[g(i, u, j) + J^*(j) \mid i, u]$. In terms of the Q-factors, the value iteration algorithm can be written as

$$Q(i, u) = \sum_{j=0}^{n-1} p_{ij}(u)(g(i, u, j) + \min_{v \in U(j)} Q(j, v)). \quad (8)$$

Simple simulation-based algorithms only involve a lookup table representation of the cost-to-go (differential costs for the average cost simulation-based policy iteration algorithm or Q-factor for the Q-learning algorithm), in the sense that a separate variable $J(i)$ is kept in memory for each state i . $J(i)$ can be calculated, for a stochastic shortest path problem for instance, as the sample mean of the cumulative cost from state i to the termination state. This cost-to-go can also be evaluated by incremental methods such as temporal difference methods.

This lookup table representation is only applicable for moderate size problems. If a given problem has very large state space, compact representation of cost-to-go need to be involved. The cost-to-go approximator can be thought as a scheme for depicting a high-dimensional cost-to-go

vectors, $\tilde{J}_\mu(i, r)$, using a lower-dimensional parameter vector. Developing a cost-to-go approximate representation involves choosing an approximate architecture, a certain functional form involving a number of free parameters, and features, which are meant to represent the most important characteristics of a given state.

Broadly, approximation architectures can be classified into two main categories: linear and nonlinear. A linear architecture is of the general form

$$\tilde{J}(i, r) = \sum_{m=0}^M r(m)\phi_m(i),$$

where $r(m)$, $m = 1, \dots, M$, are the components of the parameter vector r , and ϕ_m are fixed, easily computable functions. A common nonlinear architecture is a neural network model such as a multilayer perceptron with a single hidden layer. There are many algorithms to train the parameters; see Chapter 3 of [5] for detail.

It is often the case that the approximation architecture is too complicated for state representation and one uses instead some structural pieces to represent states. These structural pieces are called *features*, which are fed into the approximation architecture instead of the state itself. Usually, these features are handcrafted, based on the particular problem. Some examples of features include state variables, heuristic cost-to-go and/or past cost-to-go etc.

In some cases, the policy is also parameterized, as in [6] [7] [8]. And if algorithms only involve parameterized policies, they are called *actor-only* algorithms, in which the gradient of the performance, with respect to the actor parameters, is directly estimated by simulation, and the parameters are updated in the direction of improvement [7]. And if the cost-to-go function or Q-factor approximation are also involved to provide information to update the policy, they are referred to as actor-critic algorithms.

Simulation-based policy iteration algorithms and simulation-based value iteration algorithms have been successfully applied in several problems. Some of those problems are games, such as American football, Tetris and backgammon [5]; others are maintenance/repair problems [5], communication problems such as dynamic channel allocation [5] and call admission control [9], retailer management problems [10], and missile defense and interceptor allocation problems [11] etc. The features are problem dependent and in some cases, state vectors and their combinations, heuristic policies and the cost-to-go for some sub-optimal solution have been used as features. And linear architecture as well as more complicated nonlinear architectures have been applied.

Here we would like to discuss how to implement such algorithms on our fab-level decision making problem.

4 Fab-Level Decision Making MDP Model

In our IPDPM project, we proposed a Markov Decision Process (MDP) model for the highest level of the hierarchy that will yield decision support for operating the fab in each of the phases of its life cycle and include life cycle dynamics [3].

The MDP adopts an aggregate factory model for describing the state of the fab. Aggregation avoids excessive computational complexity, since a detailed factory model would have too many states. The MDP models result in policies that utilize the available information in a way that provides a trade-off between immediate and future benefits and costs, and that utilizes the fact

that observations will be available in the future (cf., e.g., [4], [12]). A policy will specify, for each possible factory state, the best actions to implement according to the objective function of the phase. Such actions include purchasing (or discarding) equipment, upgrading equipment/processes, and the allocation of equipment to product lines. Actions have costs that include the investment and operating cost and possible production shortages (from production targets), and benefits that includes increased capacity. The costs, benefits, and the system state themselves are subject to random uncontrollable events that are both exogenous and endogenous to the fab: equipment may be delivered late or may fail, the performance of newly-installed equipment is uncertain, and the market for certain chips may collapse.

In particular, the aggregate factory state is summarized by a vector of capacities $X(t)$ at time epoch t , where the components $X_{(l,i),w}(t)$ represent the capacity (measured, for example, in wafer starts per day or number of machines) of type w allocated to product l and operation i (this could be a type of sub-factory manufacturing a particular product or a type of process). Actions to be taken could be the decisions to

- (i) increase the capacity of type w by $B_w(t)$ units, possibly by the introduction of new technology; or
- (ii) switch over $V_w^{(l,i),(m,j)}(t)$ units of type w capacity from product l and operation i to product m and operation j (for example, by qualifying tools for a different process).

Randomness is explicitly modeled by the demand $d_l(t)$ for product l . The dynamics of the model will include the fact that, after the decision is made to increase capacity, there is a delay, possibly random, in the ability to fully utilize the increased capacity, and that the capacity may gradually ramp up to the expected level. The evaluation criteria include a number of factors, including costs for excess capacity and capacity shortages, cost of production, cost of converting capacity from one type of operation to another, and the cost of increasing capacity. A more precise description of the model is given below.

The state vector in period t (between time epoch t and time epoch $t + 1$) is given by $X(t) = (T_w(t), X_{(l,i),w}(t), I_l(t), l \in \mathcal{P}_t \setminus \{0\}, i \in w \setminus \{0\}, w \in \mathcal{A}_t)^T$. The actions vector in period t is $U(t) = (B_w(t), D_w(t), V_w^{(l,i),(m,j)}(t), w \in \mathcal{A}_t, l, m \in \mathcal{P}_t, i, j \in w, w \in \mathcal{A}_t)^T$, where it is assumed that $V_w^{(l,i),(m,j)}(t) = 0$ if $(l, i) = (m, j)$. At the beginning of any period, the decision maker observes the state of the system and chooses an action.

The total cost over the entire planning horizon that we want to minimize is

$$J = E \left[\sum_{t=0}^{T-1} g(X(t), U(t)) \right]$$

where

$$\begin{aligned} g(X(t), U(t)) = & \sum_{w \in \mathcal{A}_t} (C_w^a(B_w(t)) + C_w^b(D_w(t))) + \sum_{w \in \mathcal{A}_t} \sum_{\{(l,i),(m,j) | l, m \in \mathcal{P}_t, i, j \in w, (l,i) \neq (m,j)\}} C_w^c(V_w^{(l,i),(m,j)}(t)) \\ & + \sum_{l \in \mathcal{P}_t} (C_l^d(I_l(t))) + \sum_{w \in \mathcal{A}_t} \sum_{\{(l,i) \in \mathcal{P}_t \times w\}} C_w^e(X_{(l,i),w}(t)) \end{aligned}$$

We have the following state equations:

$$T_w(t+1) = T_w(t) + B_w(t) - D_w(t), \quad w \in \mathcal{A}_t, \quad (9)$$

$$X_{(l,i),w}(t+1) = X_{(l,i),w}(t) + \sum_{\{(m,j) \in \mathcal{P}_t \times w \mid (m,j) \neq (l,i)\}} (V_w^{(m,j),(l,i)}(t) - V_w^{(l,i),(m,j)}(t)), \quad (10)$$

where $l \in \mathcal{P}_t$, $i \in w$, $l \neq 0$, $i \neq 0$, $w \in \mathcal{A}_t$,

$$I_l(t+1) = I_l(t) + \min_i \left\{ \sum_{\{w \in \mathcal{A}_t \mid F_{(l,i),w} > 0\}} C_{(l,i),w} X_{(l,i),w}(t) \right\} - d_l(t), \quad l \in \mathcal{P}_t \setminus \{0\}. \quad (11)$$

The second item on the right-hand side of Equation (11) is referred to as the *throughput* in the inventory equation and gives the number of “finished wafers” of product l in period t . The operation minimizing this throughput term is referred to as the *bottleneck operation* for product l in period t .

See [3] for more details, such as notations and constraints on states.

Cost Model

The cost structure for our model is given by $\{C_w^a(x), C_w^b(x), C_w^c(x), C_l^d(y), C_w^e(x)\}$, which is rich enough to capture most cost factors in fab-level decision making. Next, we will identify major cost elements in each cost category, propose ways to estimate them, and discuss various approaches to obtain parametric values of the cost model.

The cost for additional capacity, $C_w^a(x)$, covers equipment purchase cost, equipment installation cost, equipment qualification cost, training cost, and necessary new facility cost (e.g. additional clean room). Installation cost is certain percentage of total equipment purchase cost. Qualification costs represents the cost directly involved in the initial processing of wagers to establish that the equipment is performed within specifications; and the cost includes the total labor cost involved and the production revenue lost during the period and the cost of wafer used. Training cost is counted in man-hour spent on training engineer, maintenance, operator. New facility cost is proportional to the floor space (square feet) of the clean room.

The cost for discarding capacity, $C_w^b(x)$ is equal to the residual value of the equipment, and can be handled by the so-called straight line depreciation method in accounting. If a tool is discarded within its life time (also called depreciation life), the cost for discarding is the original tool cost multiplied by a ratio which equals (life time - used time) / life time; otherwise, there is no cost for discarding.

The switch-over cost, $C_w^c(x)$, also depending on products, is a function of the tool set-up time and labor change time due to switch-over. Note there is no switch-over cost for batch tools if products are switched among that batch tool group. And there is no switch-over cost for sending capacity to reservation.

Inventory holding cost and backlogging cost are both described in $C_w^d(y)$. Inventory holding cost is related to inventory quantity, item value and length of time the inventory is carried. The cost consists of the cost of capital, variable costs of taxes and insurance on inventories, and the cost involved in storing inventory. Backlogging cost, in our case, is equal to the product's contribution margin, which is the difference between the selling price and unit production cost. Note that $C_w^d(y)$ can take market dynamics into account.

The operating cost, $C_w^e(x)$ covers the handling cost to load and unload product, tool recurring cost, labor cost for operating tools, utility cost, supplies and consumables cost, and material cost (e.g. mask), etc.

Many of the parametric values of the cost model developed are readily available or estimated in the accounting departments or other departments in a company. Labor costs, for instance, are available in the personnel department; tool purchasing costs can be obtained in the purchasing department. Inventory costs can be obtained from warehouses. The lost profit due to backlogging is not known before the total cost and income are calculated, but it can be estimated from accounting records using regression analysis. Total set-up time for each piece of equipment is typically obtained from recipes of processes. In addition, parameters can be obtained from parameters of COO models in industry.

Demand Model

We model the demand in our operational model as stochastic processes and we consider three demand patterns. It is assumed that there are three products A, B, and C, in the same product family. For example, CMOS8, CMOS10 and CMOS12 can be three products in a memory chip product family.

In pattern 1 and pattern 2, we assume that demands are independent among types of products and over periods. In pattern 3, we assume that demands for different products are correlated.

In pattern 1, we consider a short time period, with about the duration of the product's fitup ramp, and assume that the demand for product A is decreasing, the demand for product B is steady, and the demand for product C is increasing. It is meant to capture the situation that the technology for these products migrates forward from product A to product C. A special case, in which it is assumed that all demands have normal distribution and their averages are linear in time, is shown in Figure 1.

In pattern 2, we consider a longer period that ranges from the time when product A emerges to the time when product C dies out. Demand for each product has a life time. And the new product upramp compensates the downramp of the old product. The slope is not necessarily the same. If the upramp slope always greater than the downramp's, we are dealing with increasing total demand. A special case of pattern 2 with linear demand average is also shown in Figure 1.

In pattern 3, we consider correlated demand among products. Demands are given as the aggregate demands for products existing in the product family. It is assumed that the aggregate demands first increase, then become stable, and finally die out. It is meant to capture the fab life cycle dynamics. A special case with normal distribution and linear slope is shown in Figure 2.

The parameters in the patterns, such as lifetime of products, are obtained via consulting industry colleagues.

“Testbed” Example

In order to show how big our fab level decision making problem will be, we provide a “testbed” example representing a medium-size fab.

This example originates from the SEMATECH “testbed” in <http://www.eas.asu.edu/masmlab>. There are seven data sets with spreadsheets for factory, operation, products, tools, and processes etc. We chose data set 4 since it has seven products and it represents a medium-sized fab. However, there are no cost parameters in data set 4. Fortunately, data set 4 is a simple version of the ASPEN data in Factory Explorer, which gives us more information, including the cost parameters.

From the original data set, we know that three of the seven products are produced using a common process recipe A with 92 steps, and the other four products are produced using another

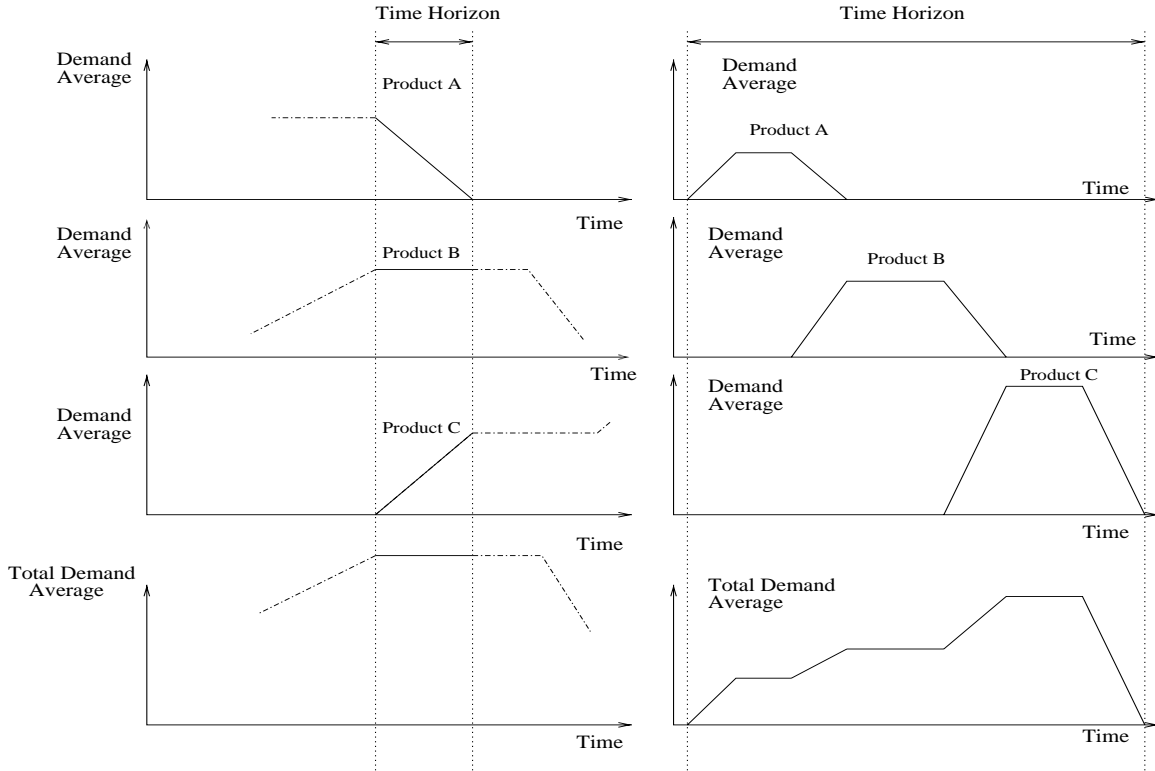


Figure 1: Demand Pattern 1 and Demand Pattern 2

common process recipe B with 19 steps. Since there are several reentry process steps, we group some reentries into one operation. For example, there are 8 clean steps and they are essentially the same operation. In this way, the 92 process steps in recipe A are aggregated into 31 operations and the 19 process steps in recipe B are aggregated into 11 operation.

So, in this example, the fab is characterized as follows. There are seven products: “A”, “B”, “C”, “D”, “E”, “F” and “G”; 137 operations: 31 operations on each of “A”, “B” and “E”, distinguished by product, and 11 operations on each of “C”, “D”, “F” and “G”; 31 tools: 3 batch tools flexible among all products, where switching-over only happens between ABE batch and CDFG batch, batch 8 tools for product A,B, and E, 12 non-batch tools flexible among A,B, and E, and 8 non-batch tools flexible among all products; operation times, which depend on the product and the tool.

If we discretize every element in the state vector into 10 values, the cardinality of the state space is 10^{175} . If we discretize every element in the control vector into 10 values, the cardinality of the control space is 10^{291} .

Note that the MDP model for the “testbed” example not only has a large state space, but it also has a large control space. Obviously, this example cannot be solved by the basic dynamic programming algorithms in the Section 2. We need to seek new approaches to solve it, such as simulation-based algorithms.

Concerning the cost model for the “testbed” example, parameters such as tool purchasing cost, depreciation life, cost per raw unit released, overhead, supplied material costs, utility cost,

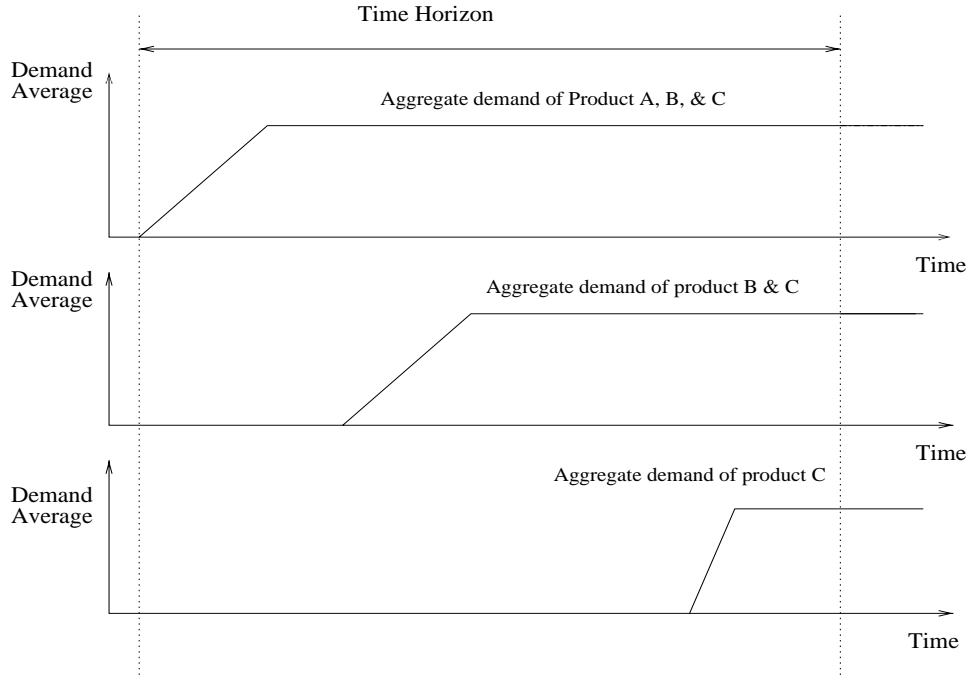


Figure 2: Demand Pattern 3

setup time and setup labor involved can be obtained from the spreadsheet for the example. Other parameters can be obtained via consulting industrial colleagues.

In the “testbed” example, the demand model for products “A”, “B”, “E” can be built following one of the three patterns in the demand model section, and demand model for products “C”, “D”, “F” and “G” can be built similarly.

5 Simulation-based Approach for Fab-Level Decision Making

We intend to use simulation-based algorithms to solve our fab-level decision making problem.

To build a compact representation for the cost-to-go function, we need to consider the following issues: what is an appropriate approximation architecture for the cost-to-go function, what are appropriate features to represent states, and how to train the approximate representation parameters? Usually, the approximation architecture is required to be rich enough to provide an acceptably close approximation of the cost-to-go function, and the features are expected to capture the most important aspects of the states. Here in our fab-level decision making problem, we have to choose between a linear architecture or a nonlinear architecture. After doing some numerical experiments, we decided that a linear architecture is enough if we assume that the components in the cost model $\{C_w^a(B_w), C_w^b(D_w), C_w^c(X_{(l,i),w}), C_w^d(I_l), C_w^e(V_w^{(l,i),(m,j)})\}$ of our problem are affine in their variables separately. In the mean time, we chose the state components and their products as features. We found that some products are not necessary to be features since the corresponding parameters are zero. We can also choose estimated cost functions from some heuristic policies as features, where the heuristic policies are obtained from solving similar or simplified capacity expansion or allocation problems. If we need to trade off control space complexity with state space complexity, we will

build a compact representation for the new states based on further numerical experiments.

We propose two heuristic ways to parameterize the policy. The first one involves dividing the original problem into several local problems, each of which relates to only one type of tool. According to (11), only bottleneck operations, the operations minimizing throughputs for products, affect the inventory level of products. Therefore, if a tool with word w , defined by a sequence of operations, does not involve any bottleneck operation, the expansion and allocation of this tool's capacity can be done locally. We call such a tool a *non-bottleneck tool (N-tool)*. For example, in the simple example, the flexible tool with $w = 013$ is an N-tool if neither operation 1 nor operation 3 is a bottleneck operation. Given a tool w , the local state is $X^{(w)} = (X_{(l,i),w}, T_{(w)})$, where $(l, i) \in \mathcal{P}_t \times w$, and the local objective function is

$$E \left[\sum_{t=0}^{T-1} [g^{(w)}(X(t), U(t))] \right],$$

where

$$\begin{aligned} g^{(w)}(X(t), U(t)) &= (C_w^a(B_w(t)) + C_w^b(D_w(t))) \\ &+ \sum_{\{(l,i),(m,j)\}} C_w^c(V_w^{(l,i),(m,j)}(t)) + \sum_{\{(l,i)\}} C_w^e(X_{(l,i),w}(t)). \end{aligned}$$

We solve these local MDP problems for all w , N-tool or not, analytically or numerically by simulation-based dynamic programming algorithms. We call the resulting policy a *local non-bottleneck policy (LN-policy)*.

On the other hand, if a tool is utilized by bottleneck operations for all products in each period, the original problem can also be solved locally. We call such a tool a *bottleneck tool (B-tool)*. In this case, the local state is $X^{(w)} = (X_{(l,i),w}, T_{(w)}, I_{(l)})$, where $(l, i) \in \mathcal{P}_t \times w$ and $l \in \mathcal{P}_t$, and the local objective function is

$$E \left[\sum_{t=0}^{T-1} [g^{(w)}(X(t), U(t))] \right],$$

where

$$\begin{aligned} g^{(w)}(X(t), U(t)) &= (C_w^a(B_w(t)) + C_w^b(D_w(t))) \\ &+ \sum_{\{(l,i),(m,j)\}} C_w^c(V_w^{(l,i),(m,j)}(t)) + \sum_{\{(l,i)\}} C_w^e(X_{(l,i),w}(t)) + \sum_l C_l^d(I_l(t)). \end{aligned}$$

We solve these local MDP problems for all w , B-tool or not, and the resulting policy is called a *local bottleneck policy (LB-policy)*.

After we solve the above local problems for each tool, we obtain two sets of policies (LN-policy and LB-policy) for each tool. The proposed heuristic policy will choose these resulting LN-policies or LB-policies with some probabilities depending on tool types (N-tool, B-tool, or neither) and operations types (bottleneck operation or not) in each period. The probabilities can be viewed as parameters to represent the policy, and the parameters can be learned by using actor-only algorithms or actor-critic algorithms on the original problem.

The second way to parameterize the policy involves dividing the original problem into two separate problems: an inventory control problem at the lower level and an expansion and allocation problem at the higher level. The connections between these two problems are the throughputs in the state equations and the cost model in the objective function. The idea is to first solve

the lower level problem with a new objective function depending on the inventory levels and the desired throughputs, and then construct expansion and allocation policies based on the desired throughputs by only considering the higher level problem with another new objective function depending on the higher level state and control components. The lower-level policy, a sequence of the desired throughputs, can be parameterized by some parameters θ_{low} . For example, if the low-level objective function has no set-up costs for the desired throughputs, a heuristic policy for the lower level can be a generalized base-stock type policy, since the optimal policy for a multi-product inventory control problem under appropriate assumptions is of this type [13]. The high-level policy, a sequence of buying, discarding and switching actions, can be parameterized by some parameters θ_{high} , following heuristic policies in the literature [14] [15].

After obtaining an appropriate compact representation for the cost-to-go function and an appropriate way to parameterize the policy, we need to use the simulation-based algorithms in the previous section.

6 Conclusion

In this paper, we discuss implementation issues of applying a simulation-based approach to a semiconductor fab-level decision making problem. The fab-level decision making problem is formulated as a Markov Decision Process (MDP). We intend to use a simulation-based approach since it can break the “curse of dimensionality” and the “curse of modeling” for the MDP with large state and control spaces. We focus on how to approximate the problem and how to parameterize the state space and control space.

Acknowledgement:

This work was supported in part by the National Science Foundation under Grant DMI-9713720, in part by the Semiconductor Research Corporation under Grant 97-FJ-491, and in part by a fellowship from General Electric Corporate Research and Development through the Institute for Systems Research.

References

- [1] S. Bermon, G. Feigin, and S. Hood, “Capacity analysis of complex manufacturing facilities,” in *Proc. of the 34th Conference on Decision and Control*, New Orleans, LA, 1995, pp. 1935–1940.
- [2] M. Zweben and M. S. Fox, *Intelligent Scheduling*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1994.
- [3] S. Bhatnagar, M. C. Fu, S. I. Marcus, and Y. He, “Markov decision processes for semiconductor fab-level decision making,” in *Proc. of the IFAC 14th Triennial World Congress*, Beijing, P. R. China, 1999, pp. 145–150.
- [4] M. L. Puterman, *Markov Decision Processes*, John Wiley & Sons, Inc., New York, 1994.

- [5] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, Massachusetts, 1996.
- [6] P. Marbach, *Simulation-Based Optimization of Markov Decision Processes*, Ph.D. thesis, MIT, 1998.
- [7] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems 12*, 2000.
- [8] R. S. Sutton, D. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in Neural Information Processing Systems 12*, 2000.
- [9] P. Marbach, O. Mihatsch, and J. N. Tsitsiklis, “Call admission control and routing in integrated services networks using neuro-dynamic programming,” submitted to *IEEE Journal on Selected Areas in Communications*, 1999.
- [10] B. Van Roy, D. P. Bertsekas, P. Lee, and J. N. Tsitsiklis, “A neuro-dynamic programming approach to retailer inventory management,” Tech. Rep., Unica Technologies, 1997, Lincoln, MA.
- [11] D. P. Bertsekas, M. L. Homer, D. A. Logan, S. D. Patek, and N. R. Sandell, “Missile defense and interceptor allocation by neuro-dynamic programming,” to appear *IEEE transactions on Systems Man and Cybernetics*.
- [12] D. P. Bertsekas, *Dynamic Programming and Optimal Control Vol 1 & 2*, Athena Scientific, Belmont, Massachusetts, 1995.
- [13] D. Beyer, S. P. Sethi, and R. Sridhar, “Stochastic multi-product inventory models with limited storage,” submitted.
- [14] S. Li and D. Tirupati, “Dynamic capacity expansion problem with multiple products: technology selection and timing of capacity additions,” *Operations Research*, vol. 42, no. 5, pp. 958–976, 1994.
- [15] S. Li and D. Tirupati, “Technology choice with stochastic demands and dynamic capacity allocation: A two-product analysis,” *Journal of Operations Management*, vol. 12, pp. 239–258, 1995.