Hierarchical Optimization of Policy-Coupled Semi-Markov Decision Processes

Gang Wang **Sridhar Mahadevan**
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48823, USA
(wanggan1, mahadeva)@cse.msu.edu

Abstract

One general strategy for approximately solving large Markov decision processes is "divide-and-conquer": the original problem is decomposed into sub-problems which interact with each other, but yet can be solved independently by taking into account the nature of the interaction. In this paper we focus on a class of "policy-coupled" semi-Markov decision processes (SMDPs), which arise in many nonstationary real-world multi-agent tasks, such as manufacturing and robotics. The nature of the interaction among sub-problems (agents) is more subtle than that studied previously: the components of a sub-SMDP, namely the available states and actions, transition probabilities and rewards, depend on the policies used in solving the "neighboring" sub-SMDPs. This "strongly-coupled" interaction among sub-problems causes the approach of solving each sub-SMDP in parallel to fail. We present a novel approach whereby many variants of each sub-SMDP are solved, explicitly taking into account the different modes of interaction, and a dynamic merging algorithm is used to combine the base level policies. We present detailed experimental results for a 12-machine transfer line, a large real-world manufacturing task. We show that the hierarchical approach is not only much faster than a "flat" algorithm, but also outperforms two well-known heuristics for running transfer lines used in many factories.

1 INTRODUCTION

Increasingly, many aspects of autonomous agents, including sensing, planning, learning, and acting are being studied using the framework of Markov decision processes (MDPs) [17]. Unlike classical methods for solving MDPs, reinforcement learning (RL) [23] is a framework for solving MDPs without any prior knowledge of the underlying state dynamics or rewards. In the RL framework, agents learn policies for acting in dynamic uncertain environments through a trial-and-error process. Although RL has had a large number of successes in recent years, a fundamental problem is that many current techniques don't scale well to larger problems.

We can categorize approaches for scaling RL into two general classes. The first approach is to construct simpler MDPs by defining temporally abstract actions, which are really policies over primitive actions. Among the various *abstraction* approaches proposed to address the scaling issue are macro-based models called "options"[24], hierarchical finite-state machine models called "HAMs" [15], and the "MAXQ" method [5].

A second general strategy for scaling RL is to decompose the original intractably large MDP. Some examples of research taking the *decomposition* approach are Dean and Lin [4], Meuleau et al. [13], Parr [14] and Singh [20]. These approaches to MDP decomposition rely more on the specific characteristics of a problem and try to "divide and conquer" the original problem. Some of them try to divide the original MDP using "boundary" states (e.g. complete or partial decoupling of navigation problems [14]), others allow subprocesses to directly affect each other through the available action set [13]. However, the coupling between the smaller MDPs in previous studies typically is "weak", allowing the divide-and-conquer approach

to be successful (e.g. [4]).

In this paper we generalize the second approach to a class of "policy-coupled" semi-Markov decision processes (SMDPs), which arise in many multi-agent domains. Although our work focuses on manufacturing given its practical importance [7, 18], examples of policy-decomposable SMDPs can be found in other multi-agent domains (e.g. robots playing soccer, two elevator systems, each of which serves half the floors etc.). The nature of the interaction is more subtle than problems previously considered: the components of each sub-SMDP, i.e. states, actions, transition probabilities and rewards, depend on policies used to solve "neighboring" sub-SMDPs. This interaction defeats the approach of solving each sub-SMDP in parallel. Yet, by analyzing the character of this interaction, we present a novel hierarchical approach that approximately solves the overall SMDP by explicitly taking into account the nature of interactions between the sub-SMDPs. Our approach is based on using an average reward SMDP Q-learning algorithm called SMART [12] for solving many variants of the constituent sub-SMDPs, and then uses a second level "dynamic merging" approach to combine these base policies into a final policy for solving the overall problem.

In Section 2, we introduce the framework of SMDPs. In Section 3, we provide an informal characterization of different ways of decomposing SMDPs, including previous work as well as the particular decomposition of SMDPs we study. In Section 4, we describe the transfer line problem in the manufacturing domain, and show how particular characteristics of this problem make it suitable for our method. Section 5 presents our hierarchical algorithm. Section 6 presents detailed experimental results of our hierarchical method applied to a large 12-machine transfer line problem, and compares it to the "flat" reinforcement learning algorithm. Section 7 summarizes the paper.

2 SEMI-MARKOV DECISION PROCESSES

We first introduce the framework of semi-Markov decision processes [17]. We also sketch an average-reward reinforcement learning algorithm for solving an SMDP, which does not assume any underlying knowledge about the transition probabilities or rewards.

A semi-Markov decision process (SMDP) is defined as a five tuple (S, A, P, R, F), where S is a set of states, A is a set of actions, P is a set of state and action dependent transition probabilities, R is a reward function, and F is a function giving probability of transition times for each state-action pair. SMDPs generalize MDPs in several ways, principally through allowing actions to be history dependent and by modeling the transition time distribution of each action.

More specifically, $P_{xy}(a)$ denotes the probability that action a will cause the system to transition from state $x \in S$ to $y \in S$. This transition probability function describes the transitions at decision epochs only, whereas the *natural process* describes the state trajectory at all times. F is a function where $F(t \mid s, a)$ is the probability that the next decision epoch occurs within t time units, after action a is executed in state s. From F, and P, we can compute Q by

$$Q(t, y \mid x, a) = P(y \mid x, a)F(t \mid x, a)$$

where Q denotes the probability that the system will be in state y for the next decision epoch, at or before t time units after choosing action a in state s, at the last decision epoch. Q can be used to calculate the expected transition time between decision epochs.

In general, the reward function for SMDPs consists of a fixed reward $k(x, a)$, resulting from an action a taken in state x, and an additional reward that is accumulated at *rate* $c(y, x, a)$ for the time the natural process remains in state y between the decision epochs. Note that the natural process may change state several times between decision epochs, and therefore, the rate at which the rewards are accumulated between decision epochs may vary.

Formally, the expected reward between two decision epochs, given that the system is in state x, and action a is chosen at the first decision epoch, may be expressed as

$$r(x, a) = k(x, a) + E_x^a \left\{ \int_0^\tau c(W_t, x, a)dt \right\} \quad (1)$$

where τ is the transition time to the second decision epoch, and W_t denotes the state of the natural process.

We have previously demonstrated a model-free average-reward algorithm (SMART) for finding gain-optimal policies for SMDPs [12]. This algorithm estimates the action value function $R^\pi(x, a)$, which is the average-adjusted sum of rewards received for the non-stationary policy of doing action a once, and then subsequently following a stationary policy π. $R^\pi(x, a)$ can be defined as

$$R^{\pi}(x,a) = r(x,a) - \rho^{\pi}\tau(x,a) + \sum_{y \in S} P_{xy}(a) \max_{b} R^{\pi}(y,b) \quad (2)$$

The temporal difference between the action-values of the current state and the actual next state is used to update the action values. In this case, the expected transition time $\tau(x,a)$ is replaced by the actual transition time, and the expected reward is replaced by the actual immediate reward. Therefore, the action values are updated as follows: [1]

$$R(x,a) \stackrel{\alpha}{\leftarrow} \left(r_{imm}(x,a) - \rho\tau + \max_{b} R(y,b) \right) \quad (3)$$

where $r_{imm}(x,a)$ is the actual cumulative reward earned between decision epochs due to taking action a in state x, ρ is the average reward, and α is a learning rate parameter. Note that ρ is actually the reward rate, and it is estimated by taking the ratio of the total reward so far, to the total simulation time.

3 A TAXONOMY OF METHODS FOR DECOMPOSING SMDPs

We now provide an informal characterization of different ways of decomposing SMDPs, including those studied in past work, and relate them to the problem class studied in our paper.

As mentioned earlier, there have been several prior methods proposed for decomposing MDPs (e.g. [4], [9], [13], [14], [20]). However, most of these focus on a specific class of MDP problems where the state space of the original problem can be divided into subsets to create smaller subproblems. In this paper we would like to broaden the concept of decomposition, where not only the state space but also other components of an SMDP, such as the action set and on a higher level, the policies for each subproblem, can be used to decompose the original problem.

First, we introduce the following concept. We term a problem $\mathcal{P}$ *SMDP-decomposable* if it satisfies the following criteria:

1. It can be described by a semi-Markov decision process.
2. It can be divided into several associated subproblems, $\mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_m$, such that:
3. Each $\mathcal{P}_i$ can be further described by a semi-Markov decision process (denoted by $\mathcal{M}_i = (S_i, A_i, P_i, R_i, F_i)$), if for each of the other subproblems $\mathcal{P}_j, j \neq i$, actions are chosen using a fixed policy $\pi_j : S_j \rightarrow A_j$.

Note that the notion of SMDP-decomposable [2] is abstractly defined, and focuses mainly on how each sub-SMDP affects each other. It doesn't specify how the original problem is decomposed to get an equivalent set of subproblems so we can utilize the solution of these subproblems to get a solution to the overall problem. As a matter of fact, for the division of the original problem to be meaningful, we have to redefine the state space, action space, state/action transition probabilities and most importantly, reward function for each subproblem. In table 1, we give various examples of SMDP-decomposable problems, classified by the nature of the interaction among sub-problems.

Navigation problems, such as the one in table 1, are typical of the class of problems suitable for *state decomposition*. In such problems, the original state space can be simply divided into subsets, each one resulting in a smaller subproblem (which are coupled by interface states, as marked by "x" in table 1). By "guessing" or "successively approximating" the expected state value of the optimal policy of the original problem ([4]) and then solving the respective subproblems, the overall policy can be constructed hierarchically from the base-level policy. In most cases of state decomposition, the internal components of each sub-SMDP, such as the available action set for each state and the transition probabilities, inherit their values from the original problem, and remain the same during the problem solving process.

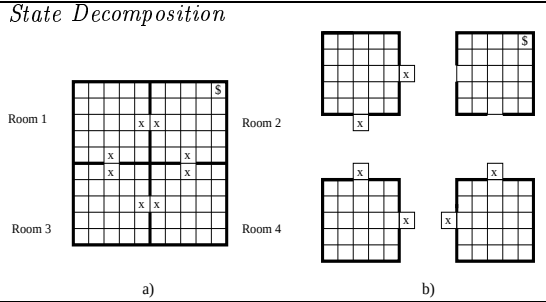
For problems approached using *action decomposition*, sub-SMDPs are coupled by the action set, such as in the air campaign planning problem [13], and the "composite" MDPs studied by Singh [20]. In problems of this category, the decision to apply a resource to a given task (e.g. an action taken in one sub-SMDP) influences the availability of that resource (action set) for other tasks. On the other hand, the transition probabilities of particular actions in one sub-SMDP are not affected by policies used to solve other sub-SMDPs. In this kind of decomposition, it is not only the state space, but also the action set through which the constituent sub-MDPs affect each other.

Finally, for problems solvable through *policy decompo-*

[1] We use the notation $y \stackrel{\alpha}{\leftarrow} x$ as an abbreviation for the stochastic approximation update rule $y \leftarrow (1-\alpha)y + \alpha x$.

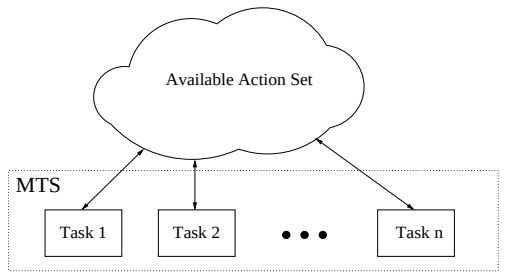
[2] In the remaining part of this paper, we view MDPs as a special case of SMDPs

State Decomposition



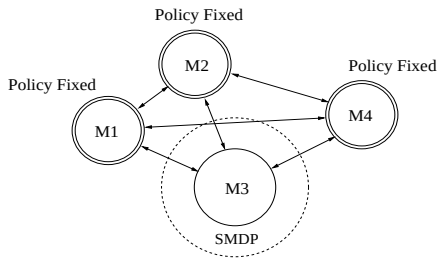
An example is a navigation problem of getting to state \$. In part b) the reward functions for all sub-SMDPs except room 2 are redefined (positive reward at "x"), but other components (e.g. actions) remain unchanged. See [4], [14], [16].

Action Decomposition



An example is the air campaign planning problem [13]. Here, the sub-SMDPs are also coupled by the set of available actions. Another example is the "factorial" composite MDP studied in [20] where the action set of the composite MDP is a proper subset of the cross product of the component action spaces.

Policy Decomposition



An example is the transfer line problem, where sub-SMPDs "strongly" interact, influencing each other's available states, actions, rewards, and transition probabilities. This problem will be discussed in detail in the next section.

Table 1: Different ways of decomposing SMDPs based on how the sub-problems interact with each other.

sition, states, actions, transition probabilities and rewards all may change when other sub-SMDPs change their policies. This makes for a more complicated coupling and therefore, requires a tricker way in approaching them.

When we examine the coupling of sub-SMDPs generated from the original problem $\mathcal{P}$, the most "weakly" coupled case occurs when the internal components of the sub-SMDP, such as the states, action sets and transition probabilities, inherit their values from the original SMDP. The coupling becomes stronger as more and more components of the sub-SMDPs are subject to change due to changes in other sub-SMDPs. Policy decomposition problems could possibly constitute the "strongest" coupling among sub-SMDPs.

In the following section we will introduce a real world instance of policy decomposition, namely the transfer line problem. In section 4 we will discuss our approach to solving this problem.

4 STOCHASTIC MODEL OF A TRANSFER LINE

The manufacturing problem we study is a discrete-part system consisting of a serial combination of distinct machines, whose overall goal is to satisfy target demand while minimizing work-in-process inventory [8]. The machines age as they are used, and are subject to breakdown and consequently need repair. Periodic maintenance can avoid unnecessary and costly repair. Transfer lines are a well-known model of factories (in particular, assembly plants) where product buffers are placed in between machines to reduce the effect of a breakdown of one machine on the overall production process. Since it is intractable to analytically compute an optimal control policy for transfer lines with more than 2 machines [8], some heuristic control methods have to be employed, such as KANBAN [1] and constant work-in-progress(CONWIP) [21]. A key disadvantage of these heuristics is that they only address the problem of minimizing work-in-process inventory. However, in reality, many other costs need to be considered, including the cost of production of parts, and the repair and preventive maintenance costs. It is all these factors that we'd like to optimize as well through solving the policy-coupled semi-Markov decision process model.

We will analyze transfer lines from the simplest case, a single machine under a fixed configuration, to the much more complicated case of multiple machines.

4.1 Single Machine Transfer Line

Before introducing transfer lines, let's briefly examine an SMDP model of a single machine (see figure 1). At each decision epoch, the single machine's action space set is $A_M = \{producing, maintenance, idling\}$. If it chooses *producing*, the machine takes supply materials from the upstream buffer and works on it until it is done and sends it to the downstream buffer. All raw materials are supplied at a *supply rate*, while all finished product in the buffer are removed from it at a *demand rate*. The action of *producing* increments a machine's running time, which when accumulated to a certain level, will lead (with high probability) to a *failure*. When the machine fails, it needs to be repaired. The failure, repair and maintenance times are all stochastic variables modeled by some distributions.

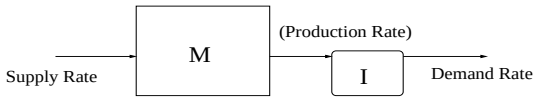


Figure 1: A single machine (M) and its inventory (I) is modeled as a semi-Markov decision process, with non-constant time actions such as "produce" or "maintain". The SMDP process is parameterized by the supply and demand rate.

If described as a SMDP, the state space S_M for a single machine transfer line can be described by a vector $< t, l >$, where t denotes the running time of the machine since last maintenance or repair; and l denotes the inventory size right after the machine. Since time is a continuous variable, for simplicity in this paper we coarsely discretize this value into intervals to turn the state space into a finite one.

The reward function R_M of the machine can be characterized by $r(x, a)$ as specified in the previous section, where $k(x, a)$ provides a fixed lump-sum reward (or cost) for each type of action (for example, a positive reward for producing, a negative cost for maintenance, and zero for idling). There is a continuous rate-cost for items that are stored in buffers. Failure and repair results in a fixed cost. These are all included in the function $c(W_t, x, a)$ in equation 1. Given a fixed demand rate and supply rate, the machine's state/action transition probabilities, $P_{xy}(a)$, for all states x and y and action a, can be specified.

Therefore, it is appropriate to say that a single machine SMDP model $\mathcal{M}$ is parameterized by three kinds of parameters: $(r_M^s, conf_M, r_M^d)$. Here r_M^s is a supply rate for machine M, r_M^d is its demand rate, and $conf_M$ includes various parameters for M's configuration such as machine failure time, repair time, producing time, buffer size, unit product reward, repair and failure cost, and finally unit cost for inventory buffer. Fortunately, these configuration parameters are fixed, and we may take them as constants for single and multiple machine transfer lines.

4.2 Transfer Line with Multiple Machines

A transfer line [8] models an assembly line operation in many factories, where a sequence of machines (or operations) is needed to make finished parts (see figure 2).

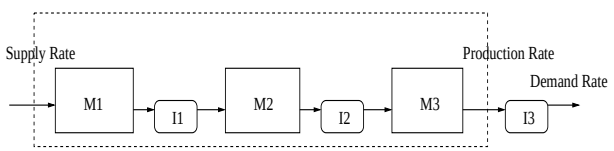


Figure 2: A transfer line with multiple machines can be viewed as a very large semi-Markov decision process, parameterized by supply and demand rate at its input and output.

The action space is the union of the expanded individual machine action spaces: $A_T = A'_{M1} \cup A'_{M2} \cup \ldots A'_{Mn}$, where A'_{Mi} is the labeled action for machine i. Considering the other machines' natural processes, when one machine is at a decision epoch, the state space of the whole transfer line is actually bigger than the cross product of the individual machine state spaces. Therefore, although the entire transfer line can still be described by a semi-Markov decision process, the state space is actually larger than the cross product of individual machine state spaces, and completely intractable.

However, if we look at the processes in a transfer line closely, we can see that the upstream machine's production rate is the downstream machine's supply rate; and the downstream machine's production rate is the upstream machine's demand rate. For example, in the transfer line of figure 3, for machine 3, the only factor among all the machines upstream that can influence it is machine 2's production rate. At the same time, since machine 3 is the last machine lined up in the transfer line, the machine's demand rate is exactly the demand rate of the whole transfer line. Therefore, we can decompose the overall "flat" semi-Markov decision process into n smaller semi-Markov decision processes which are sequentially coupled with each other. In this problem, the "contacting point" between the sub-SMDPs is each machine's production rate.

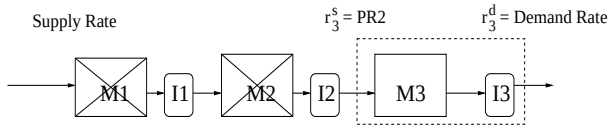


Figure 3: The overall SMDP model of a transfer line can be decomposed into a set of strongly-coupled, but significantly smaller single machine SMDPs. Here $PR2$ is production rate of machine 2. Machine 1 and machine 2 are operated with fixed policies, which makes machine 3 modelable as a semi-Markov decision process, with supply rate as $PR2$ and demand rate the demand rate of the whole transfer line.

Since each machine's production rate directly affects its neighboring machine's SMDPs, the internal dynamic character make the SMDPs more strongly coupled than that studied previously. This characteristic requires us to solve the problem in a hierarchical way, as described in the next section, and provides a much faster and better approach than "flat" optimization.

5 THE HIERARCHICAL OPTIMIZATION ALGORITHM

We now describe a hierarchical two-phase optimization algorithm, which combines an offline reinforcement learning algorithm for solving many variants of the constituent sub-SMDPs, and an online merging algorithm which dynamically recombines the learned policies.

Once the original problem is decomposed into coupled SMDPs, there are two ways to approximately solve this problem. One is to individually and simultaneously learn policies for every machine on-line by independently running n copies of the reinforcement learning algorithms together. We refer to this below as the *flat-SMART* algorithm [3]. However, for individual machine indexed i in a transfer line, its semi-Markov decision process is dependent on the machine's supply rate and demand rate, in other words, it depends on the production rate of machine $i-1$ and machine $i+1$. Therefore, if the production rates of the machine upstream and downstream are changing, the semi-Markov decision process of machine i will also vary accordingly. This makes on-line learning of an individual machine's policy in a transfer line extremely difficult and slow.

We already know that the overall semi-Markov decision process of the whole transfer line can be subdivided into a sequence of semi-Markov decision processes coupled by the actual production rates. Therefore, instead of training each machine *in-situ* while it is part of the transfer line, we train each machine in isolation using SMART for a variety of different boundary conditions, in particular, supply rates of parts coming in, and target demand rates of parts consumed by machines down the line. Once each machine is trained individually, we then optimize the transfer line as a whole, where instead of selecting each action for operating a machine, we dynamically choose between different *policies*. This algorithm is summarized in Figure 4.

This algorithm bear some similarities with other previous work. Parr [14] creates "policy caches" for subtasks before they are combined to make optimal or near optimal policies for the overall problem. In Dean and Lin's work [4], for each subproblem, a new policy is computed based on the estimation of the "boundary states" values. However, both of these systems assume that the overall problem can be divided simply by the state set, which is impossible in the transfer line problem.

- *Off-line Reinforcement Learning* $(S_{min}, S_{max}, D_{min}, D_{max}, \theta)$

  For $(s = S_{min}; s \leq S_{max}; s = s + \theta)$
  For $(d = D_{min}; d \leq D_{max}; d = d + \theta)$
  {
  Run SMART algorithm on a single machine transfer line with supply/demand rate as s/d.
  Store the obtained policy as $\Pi(s, d)$.
  } /* See Figure 6 for an example */

- *On-line Dynamic Merging* (τ)
  For each machine i in parallel, with time interval τ:
  {
  Observe over τ s_i = production rate of machine $i-1$;
  Observe over τ d_i = production rate of machine $i+1$;
  Set $\pi = \Pi(s_i, d_i)$;
  Run policy π for time τ;
  }

Figure 4: A two-phase hierarchical algorithm for optimizing a policy-decomposable transfer line SMDP.

Note we have two levels of control: the lower one is a reinforcement learning level of learning control of

[3] The SMART algorithm is a modified Q-learning method for average-reward SMDPs and is described in Section 3.

the individual machine actions. The higher level is an abstract dynamic policy merging step based on the observed supply and demand rates from machines upstream and downstream. We will refer to this technique as *hierarchical-SMART* below.

6 EXPERIMENTAL RESULTS

In this section we present detailed experimental results of the hierarchical SMART algorithm for optimization of transfer lines. We compare the performance of our algorithm to standard industrial heuristics, in particular KANBAN and CONWIP. We also compare the benefits of hierarchical decomposition with a "flat" optimization algorithm.

For the results described below, we used the following parameters. The unit (rate) cost for a single product on inventory is 2.0, the cost of failure is 40.0, and the maintenance cost is 5.0. For each product, a reward of 10 is received if it is "sold". Each machine has a failure time modeled as a normal distribution with mean of 10.0 and standard deviation of 2.0. Repair time, after each failure, is exponentially distributed with mean of 10.0. The time for maintenance is also exponentially distributed with mean of 2.0. The time to produce a product is lognormally distributed with mean of 1.0. We emphasize that our approach does not depend on the distributional or cost model, and we have conducted experiments with other cost and distributional models also.

6.1 "FLAT" OPTIMIZATION OF A 3-MACHINE LINE

We begin by comparing SMART on a three machine transfer line problem with the KANBAN and CONWIP heuristic methods. Note that this is a non-hierarchical approach, where each machine is trained while it is part of the overall transfer line. Figure 5 compares the average reward and average inventory size for the policy learned by SMART with that produced by the CONWIP and KANBAN heuristics. Each graph shows the result over 50,000 time units. In this experiment, the inventory buffer size for each machine in KANBAN is 10, and the total work-in-progress inventory in CONWIP is 11. The supply/demand rate is infinite/0.6. The results are averaged over 10 separate runs. Clearly, the average reward of the policy learned by SMART is significantly better than that achieved by CONWIP or KANBAN. This improvement is partly due to the fact that the latter heuristics do not incorporate a maintenance policy

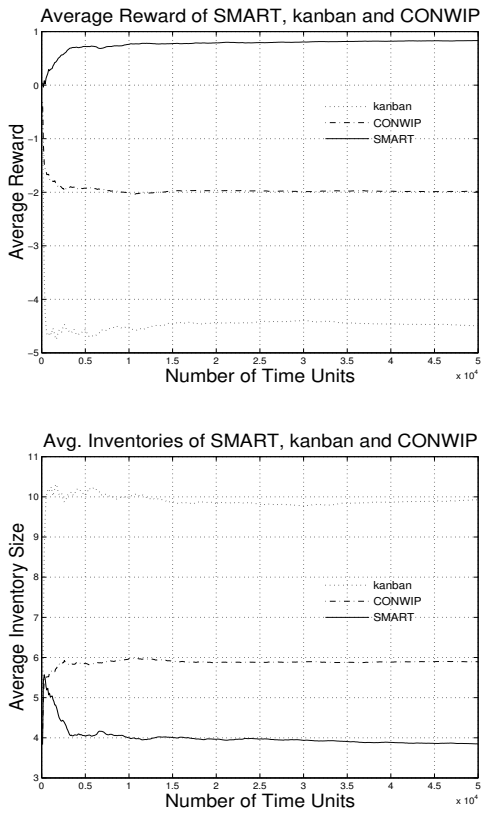


Figure 5: Comparing the average reward and inventory size learned by SMART for a three-machine transfer line with that produced by the KANBAN and CONWIP heuristics.

to minimize repair costs. Also, the policy learned by SMART also achieves a significant reduction in inventory size over KANBAN and CONWIP.

6.2 HIERARCHICAL OPTIMIZATION OF A LARGE 12-MACHINE LINE

We now turn to hierarchical optimization, where we divide the whole optimization process into two phases: an offline training phase and an online merging phase.

6.2.1 Offline Training

First, a "typical" machine is trained in isolation with different supply/demand rates. For each supply/demand rate combination, we ran SMART for 100,000 time units. The results are shown in Figure 6. Note that KANBAN performs poorly, when support rate is higher than demand rate, because it tends to fill inventory space. In such cases, SMART does a much better job at balancing the production rate along with supply and demand rate to get a higher average reward

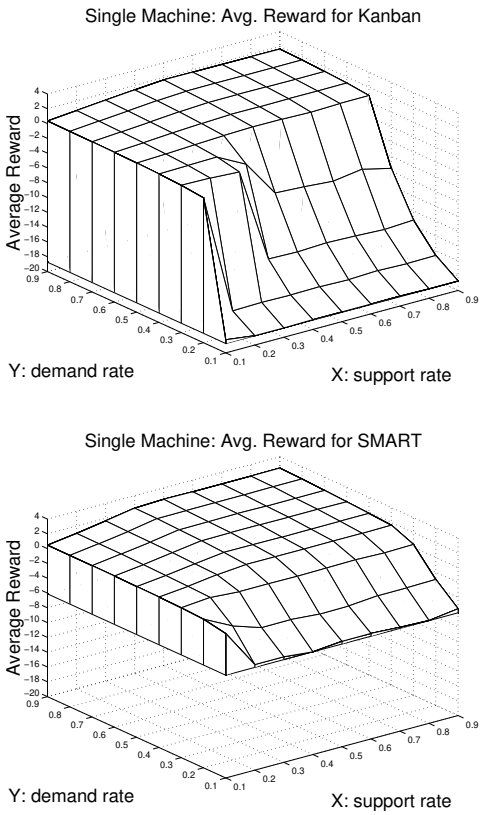


Figure 6: Average reward of KANBAN and SMART on variants of the sub-SMDPs created by varying the supply/demand rates.

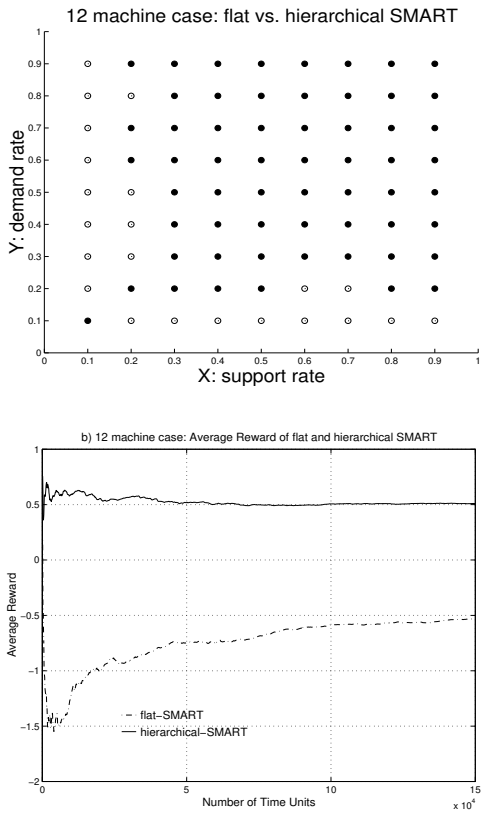


Figure 7: a) Comparison of average reward of hierarchical-SMART versus flat-SMART for a 12 machine transfer line. The solid dot means that hierarchical-SMART outperformed the flat-SMART approach. b) Average reward plot for the 12 machine transfer line for flat-SMART and hierarchical-SMART.

policy.

6.2.2 Online Merging

After training, we reinsert each machine into the line, and use the dynamic merging algorithm described in Figure 4 to compute the policy for the whole transfer line. Here, each machine is operated by selecting from one of the previously learned policies, according to its observed supply/demand rate. In this set of experiments, each machine keeps track of its supply and demand rate for the last 20 events and chooses the closest policy. For a big transfer line with 12 machines, figure 7 (a) compares the average reward of the policy learned by the hierarchical-SMART algorithm versus the flat-SMART approach, averaging over 5 runs for each approach. Figure 7 (b) is the result from a single run for both flat and hierarchical SMART. The supply/demand rate is 0.7/0.7. This experiment shows that on a larger scale optimization problem, the hierarchical SMART method results a dramatic improvement over the "flat" approach.

6.3 Transfer Line with "Bottle-neck" Machine

Here, we explore the question of what would happen if in our transfer line one machine breaks down more frequently than others. Obviously this machine cannot be treated the same as the others. Using the algorithm presented in the previous section, we train this bottle-neck machine and the "typical" machine separately, and then merge the policies as before.

In this experiment, for the bottle-neck machine in the middle of the transfer line, the mean failure time is at 3.0, instead of 10.0 time units. Figure 8 plots the average of 5 runs for each approach. From the result we can see that it is better to train the failure-prone machine separately before the policy merging step is applied. One possible consequence of this approach is that if each sub-SMDP in a policy-decomposable SMDP is very different from each other, each of them

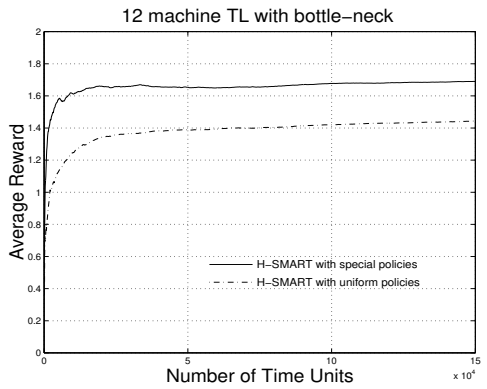


Figure 8: 12 machine transfer line with bottle-neck machine indexed 6. Ignoring and special-treating the bottle-neck machine have different results as shown in this picture.

has to be trained separately, which adds considerable computation/simulation time before the online merging phase can be used.

6.4 Varying the Temporal Intervals for Policy Merging

We also show that varying the timing interval for the abstract-level control using the policy merging step in H-SMART affects its performance. Figure 9 shows the average reward of the policy learned by H-SMART on a 12-machine transfer line with different τ values. The figure shows the average over 5 different runs for each timing interval, ranging from 10 time-events to 60 time-events. We see here clearly the more frequently we observe the demand/supply rate for each machine, the more precisely we estimate the configuration for that machine and therefore, the better the performance of hierarchical-SMART.

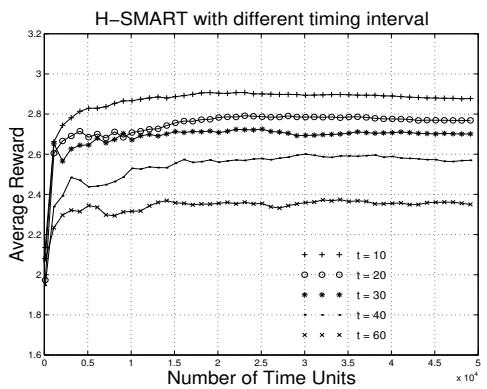


Figure 9: H-SMART with different τ values on 12 machine transfer line.

7 CONCLUSIONS AND FUTURE WORK

In this paper we proposed a novel algorithm for solving large SMDPs, which are policy-decomposable: the original SMDP can be decomposed into simpler sub-SMDPs, which strongly interact. That is, the components (states, actions, rewards, transition probabilities) for a particular sub-SMDP depend on the policies used to solve "neighboring" sub-SMDPs. We described a general algorithm (hierarchical-SMART) to optimize a real-world instance of policy-decomposable SDMPs from manufacturing, namely a transfer line. The hierarchical-SMART algorithm uses an average-reward SMDP Q-learning method to solve many variants of the individual sub-SMDPs, taking into account the modes of interaction among them, and then uses a dynamic policy merging strategy at the abstract level to compute a policy for the entire transfer line. The experimental results showed that this approach is superior to well-established heuristics for operating transfer lines, as well as to using a "flat" approach of learning to control each machine in parallel.

Many issues remain unexplored in this research. There are many other examples of policy-coupled SMDPs in manufacturing, such as jobshops and flowshops, which are possibly even more challenging than transfer lines. As another example, consider the RoboSoccer task where different agents learn in a non-stationary environment to play soccer. If we view each agent as a sub-SMDP, then the various sub-SMDPS are clearly policy-coupled: each agent can be trained to reliably play a certain "role" only when all the other agents use fixed policies. In fact, recent work has employed training sessions of this type, e.g. see Stone [22]. Another issue that remains to be investigated is policy-coupled SMDPs with deeper abstraction hierarchies, such as in MAXQ [5]. The taxonomy of decomposition methods that we proposed is somewhat intuitive. Clearly, a more formal characterization would be useful. Finally, a general algorithm for solving arbitrary forms (or even special classes) of policy-coupled SMDPs would be a significant step forward.

Acknowledgement

This research is supported in part by an NSF CAREER Award Grant No. IRI-9501852, and by a Michigan State University (MSU) AURIG grant.

References

[1] Berkley, B. J. 1992. "A review of the KANBAN production control research literature" in *Production and Operations Management 1(4)*, 393-411.

[2] Bonvik, A. M. Couch, C. Gershwin, S. 1996. "A Comparison of Production-Line Control Mechanism". Accepted for *International Journal of Production Research*.

[3] Crites, R. Barto, A. 1996. "Improving elevator performance using reinforcement learning" in *Neural Information Processing System (NIPS)*.

[4] Dean, T. Lin, S.H. 1995. "Decomposition Techniques for Planning in Stochastic Domains", in *Proceedings of IJCAI'95*, Montreal, 1995.

[5] Dietterich, T. G. 1998. "The MAXQ method for hierarchical reinforcement learning". Proceedings of 1998 International Conference on Machine Learning.

[6] Gershwin, S., Caramanis, M., Murray, P. 1988. "Simulation Experience with a Hierarchical Scheduling Policy for a Simple Manufacturing System", in *Proceedings of the 27th IEEE Conference on Decision and Control*, Austin, Texas.

[7] Gershwin, S. 1987. "A Hierarchical Framework for Manufacturing System Scheduling: A Two-Machine Example." in *Proceedings of the 26th IEEE Conference on Decision and Control*, Los Angeles, CA.

[8] Gershwin, S. 1994. *Manufacturing System Engineering.* Prentice Hall.

[9] Kaelbling, L. P., 1993. "Hierarchical Reinforcement Learning: Preliminary Results," in *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[10] Law, A. Kelton, W. 1991. *Simulation Modeling and Analysis.* New York, USA: McGraw-Hill.

[11] Mahadevan, S. 1996. "Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results" in *Machine Learning.*, 22, 159-195.

[12] Mahadevan, S., Marchalleck, N., Das, T., and Gosavi, A. 1997. "Self-improving factory simulation using continuous-time average reward reinforcement learning" in *Proceedings of the Fourteenth International Machine Learning Conference.*, D. Fisher (editor), Morgan Kaufmann. 202-210.

[13] Meuleau, N., Hauskrecht, M., Kim, K, Peshkin, L., Kaelbling, L., Dean, T., Boutilier, C. 1998. "Solving Very Large Weakly Coupled Markov Decision Processes", in *Proceedings of the Conference on Uncertainty in AI*, 1998.

[14] Parr, R. 1998. "Flexible Decomposition Algorithms for Weakly Coupled Markov Decision Problems" in *Proceedings of the Conference on Uncertainty in AI*, 1998.

[15] Parr, R. and Russell, S. 1998. "Reinforcement Learning with hierarchies of machines" in *Advances in Neural Information Processing Systems (NIPS).*

[16] Precup, D., Sutton, R. S. 1997. "Multi-time models for temporally abstract planning", in *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference*, Denver, Colorado. MIT Press.

[17] Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc.

[18] Sethi, S., and Zhang, Q. 1994. *Hierarchical Decision Making in Stochastic Manufacturing Systems.*, Birkhauser.

[19] Singh, S. Bertsekas D. 1996. "Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems" in *Neural Information Processing Systems (NIPS)*, 1996.

[20] Singh S, Cohn D. 1998, "How to Dynamically Merge Markov Decision Processes", In NIPS 11.

[21] Spearman, M. L., D. L. Woodruff, and W. J. Hopp. 1990. "CONWIP: a pull alternative to KANBAN" in *International Journal of Production Research 28(5)*, 879-894.

[22] Stone, P., "Layered Learning for Multi-Agent Systems", Ph.D. thesis, Carnegie Mellon University, December 1998.

[23] Sutton, R., Barto, A. 1998. *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA. A Bradford Book.

[24] Sutton, R., Precup, D., and Singh, S. 1998, "Intra-option Learning about Temporally Abstract Actions", Proceedings of the 15^{th} International Conference on Machine Learning (IMLC-98), pp. 556-564.