To appear in Proc. AI and Manufacturing Research Workshop,
AAAI, June 1996

# Over-Constrained Scheduling using Dynamic Programming

**Edward Sobiesk**
Department of Computer Science
University of Minnesota
200 Union Street SE, Room 4-192
Minneapolis, MN 55455
sobiesk@cs.umn.edu

**Kurt Krebsbach**
Honeywell Technology Center
Automated Reasoning Group
3660 Technology Drive
Minneapolis, MN 55418
krebsbac@src.honeywell.com

**Maria Gini**
Department of Computer Science
University of Minnesota
200 Union Street SE, Room 4-192
Minneapolis, MN 55455
gini@cs.umn.edu

**Abstract**

In this paper, we demonstrate the use of stochastic dynamic programming to solve over-constrained scheduling problems. In particular, we propose a decision method for efficiently calculating, prior to start of execution, the optimal decision for every possible situation encountered in sequential, predictable, over-constrained scheduling domains. We present our results using an example problem from Product Quality Planning.

## Introduction

In this paper, we demonstrate the use of stochastic dynamic programming to solve over-constrained scheduling problems. In particular, we propose a decision method for efficiently calculating, prior to start of execution, the optimal decision for every possible situation encountered in sequential, predictable, over-constrained scheduling domains.

Over-constrained scheduling is a problem frequently encountered in the manufacturing industry. The general situation can be described as having a time-constrained goal that requires the execution of a set of tasks that have priorities, required durations, and ordering constraints[1]. In the "over-constrained" situation, the amount of time available before the deadline is less than the sum of the tasks' durations. Decisions must, therefore, be made about how to schedule the inadequate amount of time available to optimize the results. As also frequently happens, because task execution is nondeterministic, we may find that in the middle of executing our tasks, we are either ahead or behind where we had planned to be. We would like our decision method to tell us immediately what is the best action to take from any of these unexpected states.

For our current model of the over-constrained scheduling domain, we assume that there is value in partially completing tasks, but because most tasks are more coherent if executed entirely in sequence, we assume tasks are preemptable but not continuable. We also assume domains where all tasks can be anticipated prior to execution and are executed sequentially.

The work presented here builds on the earlier work of (Krebsbach 1993) who introduced the use of stochastic dynamic programming to construct sensor schedules.

The ability of our method to allow immediate and continuous update of the optimal decision to make from any situation is a distinct advantage it possesses over other decision methods (PERT/CPM, linear programming, etc) (Anderson, Sweeny, & Williams 1994). Our automated method precomputes the optimal decision from every possible state, whereas other methods require recomputation when the current situation no longer matches the expected situation of the plan. Other decision methods also do not generally deal with the problems of over-constrained scheduling.

## A Motivating Example: Over-Constrained Product Quality Planning

We now present an instance of the over-constrained scheduling problem using as an example Product Quality Planning. Suppose you work for a small company and are assigned a Product Quality Planning Project for a new generation of an existing product. Because you will be the only person working on the project, sequential (nonconcurrent) execution is required. Let the following be the general tasks, priorities (higher is

[1] We define the term *ordering constraint* to mean that one task is only of value if executed before the other. Therefore, a *before task* is no longer considered for execution if the *after task* has already been executed. We do not, though, consider execution of the *before task* as a prerequisite for executing an *after task*.

more important), durations, and necessary ordering of tasks:

| | Task | Priority | Duration |
|---|---|---|---|
| 1. | Plan and Define Program | 1 | 2 |
| 2. | Product Design and Development | 3 | 5 |
| 3. | Process Design and Development | 3 | 4 |
| 4. | Product and Process Validation | 2 | 2 |
| 5. | Feedback Assessment and Corrective Action | 1 | 2 |

Figure 1: Tasks for the Product Quality Planning example problem. Durations are in months.

Many methods could be used to produce the same general schedule when the required time is available:

Month 1-2 : Plan and Define Program
Month 3-7 : Product Design and Development
Month 8-11 : Process Design and Development
Month 12-13: Product and Process Validation
Month 14-15: Feedback Assessment and Corrective Action

Now suppose you are told that because of the competitive race to market your product, instead of the required 15 months, you will only be allotted 12 months. How do you decide how much time to spend on each task? Most of the time, you would have to use intuition and experience to judge what to do. You would like to accomplish as much as possible of the most important tasks. However, to maximize your productivity, you would need to take into account how much of each task can be achieved in each time unit spent on it. Deciding exactly on the optimal allocation is not trivial. Based on the new 12 month constraint, you would want a decision method that would give you the new optimal schedule of:

Month 1 : Plan and Define Program
Month 2-6 : Product Design and Development
Month 7-9 : Process Design and Development
Month 10-11: Product and Process Validation
Month 12 : Feedback Assessment and Corrective Action

Now, what if at five months before the deadline, you have completed all of tasks 1 and 2, but no part of the remaining three tasks? You would want your decision method to give you the optimal schedule of:

Month 1-3: Process Design and Development
Month 4 : Product and Process Validation
Month 5 : Feedback Assessment and Corrective Action

The above examples illustrate exactly the ability that stochastic dynamic programming possesses. We will now explain the technique that we use, and how our method guarantees that all decisions are optimal.

In the remainder of the paper, we will first give a description of the general methodology of stochastic dynamic programming. We will then cover its specific implementation for the over-constrained scheduling domain. Next, we will show the results of our method and reveal the statistics and computation that resulted in the decisions for our Product Quality Planning example. We will conclude the paper with a brief discussion of the strengths and limitations of our method, and related work.

## Stochastic Dynamic Programming

A *decision method* tells an agent what decision to make at each decision point. Constructing a sequence of decisions can be viewed as a *sequential decision problem* (SDP)[2] (Boddy 1991b; Hillier & Lieberman 1980). Stochastic dynamic programming (Ross 1983; Howard 1960) is a solution method for problems of this sort. We now formally describe the components of stochastic dynamic programming.

### Fundamental Characteristics

- The problem can be divided into **stages**. Each stage has a number of associated **states** with a **decision** required for every state. Each state contains information sufficient for making the decision. Let $\mathcal{S}$ be the set of **states** the system can assume and $\mathcal{D}$ be the set of **decisions** that can be made at each decision point.

- The effect of a decision on a state is to transform the current state into a state in the next stage. Let $\phi : \mathcal{S} \times \mathcal{D} \rightarrow \mathcal{S}$ be a **stochastic function** mapping from the current state and a decision to a set of states in the next stage according to a given probability distribution.

- Given the current state, **optimal decisions** for states in the remaining stages can be considered **independently** of the decisions adopted in previous stages. This crucial property means that knowledge of the current state of the system conveys all the information about its previous behavior necessary to

[2]Sequential decision problems are a special case of so-called *multi-stage decision problems*(Bellman 1957).

make optimal decisions from that stage on. It is sometimes referred to as the *principle of optimality* (Bellman 1957), and is a special case of the *Markovian property*, which says that the conditional probability of any future event, given any past event and the present state is independent of the past event and depends only upon the present state of the process. Let $G : \mathcal{S} \times \mathcal{D} \rightarrow \mathcal{R}$ be a **gain function** mapping from the current state and a decision to the real numbers[3]. Conceptually the gain function represents the expected execution gain that a given decision in a given state implies. Of course, the resulting solution is highly dependent on our choice of gain function, which must take into account the expected gains resulting from its decision. The gain function can be thought of as a numerical representation of the relative advantages and disadvantages of one decision over another given a particular state.

### Computational Method

1. The solution procedure begins by finding an optimal decision for all possible states of the **final** stage. This is often trivial.

2. A **recursive function** is available which identifies an optimal decision for each state at stage $i$, given a one-step local gain function and already computed optimal decisions for each state at the successor stage $(i + 1)$. For our purposes, finding an optimal (one step) decision from a state in stage $i$ thus begins with computing the local gain of each legal decision for that state. It then finds the global gain for each of these possible decisions by adding the decision's local gain to the accumulated (backed up) global gain from the relevant successor state at stage $i + 1$. It chooses the decision which maximizes this overall global gain towards the goal state. All other $(state, decision)$ pairs can be discarded because the principle of optimality holds. This is where a great deal of computation and storage is saved by using dynamic programming because only the optimal states are stored and referenced for future computation. The function that identifies an optimal decision for each state at stage $i$ is traditionally expressed as:

$$f_i^*(s) = max\{g_{sx_i} + f_{i+1}^*(x_i)\}$$

where

$f_i^*(s)$ is the global gain for state $s$.

$f_{i+1}^*(x_i)$ is the global gain for the state at stage i+1 entered by making decision $x_i$ at stage i.

$x_i$ is the decision being considered at decision point $i$.

$g_{sx_i}$ is the local gain of the transition from state $s$ at stage $i$ to the appropriate successor state in stage $i + 1$ by choosing action $x_i$.

An optimal decision method consists of finding the value of $x_i$ that maximizes $f_i^*(s)$.

3. Using this recursive relationship, the solution procedure moves backward stage by stage, each time determining an optimal decision for the states at that stage. This optimizes the sequence of decisions from that state forward. This continues until it finds an optimal decision at the initial stage. Note that intuitively this entails storing optimal decisions in a table that can be referenced by a state. While it is possible to proceed forward from the first to final stage for some dynamic programming problems, it is generally not possible when stages correspond to time periods. This is because possibly optimal sequences of decisions might be discarded if they look more costly early on when some commitments must be made about optimal subsequences of decisions. However, this is never a problem when working backward from the final to initial stage. As long as the principle of optimality holds, an optimal decision for each state in the current stage can be assumed optimal regardless of how that state is reached.

## Over-Constrained Scheduling Decision Table Generation

The purpose of this section to explain our specific application of stochastic dynamic programming to the over-constrained scheduling domain.

The execution of the application requires several user inputs. It is important to note that our method's guarantee of optimality is conditionalized on the accuracy of these user inputs. The inputs required include:

- *Task Names.*
- *Task Priorities.* The algorithm is designed such that higher priorities are more important.
- *Task Durations.* Must be all in the same time unit.
- *Estimated Percentages Completed per Time Period.* This input is required for each task. It is a vector of the fractions of the task that the user believes will get accomplished with each sequential time period spent on the task. As an example, for the Product Quality Planning task of Plan and Define Program, we chose a task duration of two and our *EstPercComp* vector was (4/5 1/5). This is equivalent to

[3]This is traditionally called the *reward function* in operations research.

saying that we believe 80% of the task will be accomplished during one month's work, and that the remaining 20% of the task will be accomplished working on it a second month. Note that the *EstPercComp* vector fractions must sum to one. A major strength of our decision method is that by using the *EstPercComp* vector to calculate the anticipated local gain of a task for each time period of execution, we allow the user a nonlinear method of expressing anticipated task accomplishment.

- *Ordering Constraints.* Any ordering constraints that exist between tasks. The default order of execution is by task priority from highest to lowest.
- *Number of Stages.* The number of time periods until the deadline.
- *Time Period Type.* The time period unit being used. For our example, we chose months.

Figure 2 contains the algorithm we use for constructing an optimal decision table in the over-constrained scheduling domain. The algorithm is fairly straightforward. The outer loop moves the process backward from the final stage to the initial stage. All possible states at each stage are generated and stored in the list $S_i$. For each of these states, we compute the expected one-step gain of making each possible decision by multiplying the appropriate *EstPercComp* by the *Task Priority*. We add these one-step gains to the accumulated global gains from the projected next stage states. States are projected using the function $\phi$ defined earlier. We store the decision which yields the maximum *GlobalGain* for the state. The general form of the gain function describing the expected gain from making decision $d$ in state $s$ is simply:

$$LocalGain(s,d) = EstPercComp(s,d) * TaskPriority(d)$$

$$GlobalGain(s) = max_d[LocalGain(s,d) + GlobalGain(\phi(s,d))]$$

The specific part of the *EstPercComp* vector used in the calculation of *LocalGain* is based on how many time periods we have worked so far on the task. For instance, for the task Plan and Define Program, the *EstPercComp* vector is (4/5 1/5) and the *Task Priority* is 1. So, *LocalGain* for the first time period spent working on the task would be 4/5 * 1 = 4/5.

We utilize the combination of a state with the number of stages remaining as the index to the optimal decision table. So the index for the initial situation in the over-constrained 12 month Product Quality Planning example would combine the number of stages remaining with the *Task Duration* for each task. So the index would be (12 2 5 4 2 2).

```
For i ← Stages downto 1
    begin
    ;; make a list of all possible states for stage i
    S_i ← PossibleStates(i)
    ;; for each state find optimal decision
    For each s in S_i
        begin
        ;; SlackTime is default decision
        Decision(s) ← SlackTime
        GlobalGain(s) ← 0
        ;; for each legal decision for the state
        For each legal decision d for s
            begin
            LocalGain(s,d) ← EstPercComp×
                TaskPriority
            GlobalGain(d) ← LocalGain(s,d)+
                GlobalGain(φ(s,d))
            If GlobalGain(d) > GlobalGain(s)
                begin
                GlobalGain(s) ← GlobalGain(d)
                Decision(s) ← d
                end
            end
        Store GlobalGain(s)andDecision(s)
            as optimal decision and gain at
            Table Index(i,s)
        end
    end
```

Figure 2: Algorithm for generating the optimal decision table.

```
For i ← 1 to Stages
    begin
    Update the CurrentState
    d ← GetStoredDecision(i, CurrentState)
    Execute(d)
    end
```

Figure 3: Utilizing the optimal decision table during execution.

Figure 3 shows how the optimal decision table can be used at execution time to make the proper state-dependent, optimal decisions. To avoid burdening the user with determining what state he is in, an interface will be designed to take information from the user

about how many stages remain and how much of each task he estimates is completed. Based on the user's *EstPercComp*, the program will then calculate how many time periods are left to complete each task, index the appropriate state, and output the optimal decision.

## Results

The method we have described has been fully implemented and applied to many problems in the over-constrained scheduling domain.

Figures 1 and 4 show the input used for our Product Quality Planning example problem. Here is the output of our scheduler for the original 15 month unconstrained scheduling problem:

| | | |
|---|---|---|
| Month 1 : | Plan and Define Program | Gain: 10.0 |
| Month 2 : | Plan and Define Program | Gain: 9.2 |
| Month 3 : | Product Design and Development | Gain: 9.0 |
| Month 4 : | Product Design and Development | Gain: 8.1 |
| Month 5 : | Product Design and Development | Gain: 7.2 |
| Month 6 : | Product Design and Development | Gain: 6.6 |
| Month 7 : | Product Design and Development | Gain: 6.3 |
| Month 8 : | Process Design and Development | Gain: 6.0 |
| Month 9 : | Process Design and Development | Gain: 5.7 |
| Month 10: | Process Design and Development | Gain: 4.5 |
| Month 11: | Process Design and Development | Gain: 3.3 |
| Month 12: | Product and Process Validation | Gain: 3.0 |
| Month 13: | Product and Process Validation | Gain: 1.6 |
| Month 14: | Feedback Assessment and Corrective Action | Gain: 1.0 |
| Month 15: | Feedback Assessment and Corrective Action | Gain: 0.3 |

The generated table of optimal decisions is now used to find the optimal decision given that only 12 months are left and nothing has been done yet. The schedule is:

| | | |
|---|---|---|
| Month 1: | Plan and Define Program | Gain: 9.2 |
| Month 2: | Product Design and Development | Gain: 8.4 |
| Month 3: | Product Design and Development | Gain: 7.5 |

| Task | EstPercComp | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| 1. Plan and Define Program | 0.8 | 0.2 | | | |
| 2. Product Design and Development | 0.3 | 0.3 | 0.2 | 0.1 | 0.1 |
| 3. Process Design and Development | 0.1 | 0.4 | 0.4 | 0.1 | |
| 4. Product and Process Validation | 0.7 | 0.3 | | | |
| 5. Feedback Assessment and Corrective Action | 0.7 | 0.3 | | | |

Figure 4: Estimated percentages of task completed per month.

| | | |
|---|---|---|
| Month 4 : | Product Design and Development | Gain: 6.6 |
| Month 5 : | Product Design and Development | Gain: 6.0 |
| Month 6 : | Product Design and Development | Gain: 5.7 |
| Month 7 : | Process Design and Development | Gain: 5.4 |
| Month 8 : | Process Design and Development | Gain: 5.1 |
| Month 9 : | Process Design and Development | Gain: 3.9 |
| Month 10: | Product and Process Validation | Gain: 2.7 |
| Month 11: | Product and Process Validation | Gain: 1.3 |
| Month 12: | Feedback Assessment and Corrective Action | Gain: 0.7 |

The table is used again to find the optimal decision given that five months are left and only the first 2 tasks have been completed. Here is the schedule:

| | | |
|---|---|---|
| Month 1: | Process Design and Development | Gain: 4.8 |
| Month 2: | Process Design and Development | Gain: 4.5 |
| Month 3: | Process Design and Development | Gain: 3.3 |
| Month 4: | Product and Process Validation | Gain: 2.1 |
| Month 5: | Feedback Assessment and Corrective Action | Gain: 0.7 |

## Strengths, Limitations and Assumptions

As we have demonstrated throughout the paper, our method has many strengths. The most important of these is its ability to precompute the optimal decision for all possible states. For reusable domains, this will have the benefit of a one time computation that can then be used by an executing agent. Another major advantage of our method is its flexibility due to the user determining the input statistics. While this puts the burden of expertise on the user, it does allow for the adaptability of our method to numerous diverse domains. Although our Product Quality Planning example utilized month time periods, we believe our method would be especially useful in over-constrained problems involving hour time periods. The automation of our method provides great speed, and its guarantee of optimality would be extremely beneficial at times when stress and tiredness may impair human judgement and intuition. Finally, since the optimal decisions are computed and stored using stochastic dynamic programming, this method has been shown to be polynomial in both space and time (Krebsbach 1993).

The two major limitations of our method are the size of the generated table of optimal decisions and the requirement of our method to have all task information input prior to start of execution. We believe, though, that many important domains exist for which our method is applicable.

As we stated in the introduction, our current implementation makes a few assumptions. We schedule only sequential tasks. We assume no task is absolutely essential, and so tasks can be completely skipped. The priority of the tasks, and how much of each task can be accomplished in each time unit determine how much time to devote to each task. We assume that the goal requires a set of known tasks which can be anticipated from the beginning and will not change throughout execution. This assumption is essential for precomputing all possible states and optimal decisions. We assume there is value in a partially completed task (an 80% solution on time is better than a perfect solution too late), so we allow tasks to be preempted. However, once a task is preempted we do not allow it to be resumed. Finally, we assume the ability to measure the accomplishment of tasks in a discrete manner. This allows us to index the current progress towards the goal by the status of how much has been accomplished on each task so far.

## Related Work

Decision-theoretic methods have been used for a wide variety of optimization and control problems, including those related to planning and sensing (Bellman 1957; Boddy 1991b). Boddy proposes the use of dynamic programming for constructing anytime algorithms (Boddy 1991a). Hager and Mintz (1991) have proposed methods for sensor planning based on probabilistic models of uncertainty. Goodwin and Simmons (1992) use a decision-theoretic approach to incorporate the achievement of a new goal into a partially executed plan, and Chrisman and Simmons employ Markov Decision Processes in order to handle a significant amount of uncertainty in the outcomes of actions (1991). Hansen (Hansen 1994) incorporates sensing costs in the framework of stochastic dynamic programming. Wellman and Doyle (Wellman & Doyle 1992) propose modular utility functions for decision-theoretic planning. Modular functions allow specifying preferences, and composition methods allow combining them.

The method we propose can be considered similar to universal planning (Schoppers 1987) and to "tree plans" (Nilsson 1994). We generate a complete set of schedules that will work for all possible contingencies. The advantage of our method is that we guarantee the optimality of all the decisions, while universal planning is more concerned with having a plan for all contingencies, not necessarily the optimal plan.

## Conclusion

We have demonstrated the use of stochastic dynamic programming to solve over-constrained scheduling problems. In particular, we proposed a decision method for efficiently calculating, prior to start of execution, the optimal decision for every possible situation encountered in sequential, predictable, over-constrained scheduling domains. The chief idea behind our method is that optimal decisions are calculated prior to start of execution and stored in a table. The actual state of the world is then used as an index into the table during execution.

## References


Anderson; Sweeny; and Williams. 1994. *Management Science.* West Publishing Co, 7th edition.

Bellman, R. 1957. *Dynamic Programming.* Princeton, NJ: Princeton University Press.

Boddy, M. 1991a. Anytime problem solving using dynamic programming. In *Proceedings of AAAI-91.*

Boddy, M. 1991b. Solving time-dependent problems: A decision-theoretic approach to planning in dynamic environments. Tech Report CS-91-06, Brown University, Department of Computer Science.

Chrisman, L., and Simmons, R. 1991. Sensible planning: Focusing perceptual attention. In *Proceedings of AAAI-91.*

Goodwin, R., and Simmons, R. 1992. Rational handling of multiple goals for mobile robots. In *Proceedings of the First International Conference on AI Planning Systems*, 70–77.

Hager, G., and Mintz, M. 1991. Computational methods for task-directed sensor data fusion and sensor planning. *International Journal of Robotics Research* 10:285–313.

Hansen, E. A. 1994. Cost-effective sensing during plan execution. In *Proc. Nat'l Conf. on Artificial Intelligence*, volume 2, 1029–1035.

Hillier, F. S., and Lieberman, G. J. 1980. *Introduction to Operations Research.* San Francisco, CA: Holden-Day, Inc., 3rd edition.

Howard, R. A. 1960. *Dynamic programming and Markov processes.* MIT Press and John Wiley & Sons.

Krebsbach, K. 1993. Rational sensing for an AI planner: A cost-based approach. Ph.D. dissertation, University of Minnesota.

Nilsson, N. 1994. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research* 1:139–158.

Ross, S. 1983. *Introduction to Stochastic Dynamic Programming.* New York, NY: Academic Press.

Schoppers, M. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1039–1046.

Wellman, M., and Doyle, J. 1992. Modular utility representation for decision-theoretic planning. In *International Conference on AI Planning Systems*, 236–242.