Managing Operations through the Control of Information

Warren B. Powell

January, 2000

Department of Operations Research and Financial Engineering
Princeton University
Princeton, NJ 08544

Consider one of our standard problems in operations research. Perhaps we are moving product through a global supply chain to satisfy a customer. Or we might be planning the movement of our trucks and aircraft, with their associated drivers and pilots, to move small packages overnight in response to that day's demands from the internet. If we are feeling brave, we might be interested in taking on the task of planning, in real time, the problem of managing the locomotives or crews of a major freight railroad, moving thousands of trains between hundreds of locations.

In all of these applications, we would usually end up formulating a model that looks something like:

$$\min \sum_{t=0}^{T} \sum_{k=1}^{K} c_t^k x_t^k \tag{1}$$

subject to:

$$A_{t-1}^k x_{t-1}^k + B_t^k x_t^k = b_t^k \tag{2}$$

$$\sum_{k=1}^{K} x_t^k \leq u_t \tag{3}$$

$$x_t^k \geq 0 \tag{4}$$

In our "little" optimization problem, we have chosen to capture that our problem is probably defined over time, where equation (2) couples different time periods together. And we are probably tracking the flows of different types of commodities, which are bundled somewhere with an equation looking like (3). Of course, equation (4) is our omnipresent nonnegativity constraint (we might also need to add integrality constraints).

For the types of large scale problems with which we opened our discussion, this optimization problem is very hard to solve. Even with modern workstations and the major advances in linear programming algorithms, the truly large scale multicommodity flow problems remain a challenge. Making the

problem truly intractable is the likely presence of many integer variables. Needless to say, this problem remains a very active area of research.

But perhaps what makes this problem *really* hard is that we do not even know the values for the constraint matrices *A* and *B* (where we have to capture uncertain travel times), the cost vector *c* (where we have to capture soft preferences for different types of activities), and the resource constraint *b*, where we capture the future demands from customers (see Sen and Higle, 1999, for an overview of optimization under uncertainty).

Decomposing large problems

In actual large-scale operations, it is customary to break up a large problem into numerous small problems. For our purposes, it is useful to think of a set Q as representing the set of people

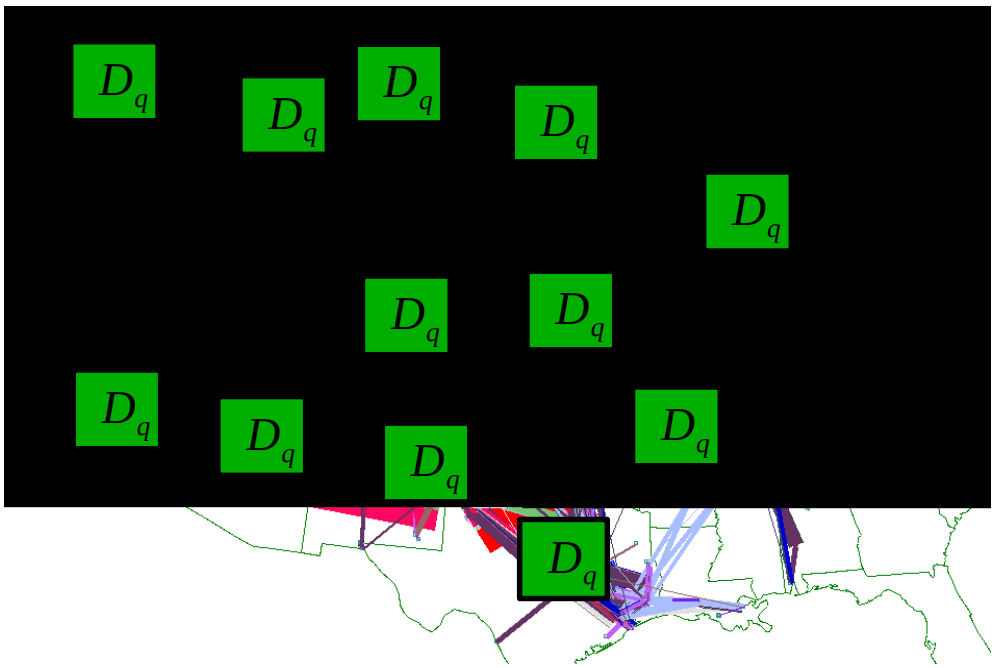

Figure 1

Illustration of the decomposition of a large scale operation into a set of different subproblems. Each subproblem q is comprised of a set of decisions D_q.

making decisions, and let $q \in Q$ be a particular decision maker. Further let D_q be the set of decisions that are made by decision maker q. Figure 1 illustrates a set of activities for a large scale operation (our example is drawn from a major railroad) into a set of subproblems. (Our notation in this paper is based on Powell and Shapiro, 2000).

When we decompose a problem into subproblems, it is important to capture who impacts whom. A useful bit of notation (which we need later) is the *forward reachable set*, denoted M_q. This is the set of subproblems that are *directly* impacted by decisions made in subproblem *q*.

Normally, we motivate the design of a set of subproblems based on the mathematical structure of the original problem. In the real world, a subproblem is somewhat more tangible. In fact, a good example of a subproblem is shown in figure 2. For convenience, we will refer to our subproblem as "Harry." We observe that Harry is doing what a lot of humans do in operations – he is looking at a sheet of paper and talking on the phone. Both of these represent a form of communication that does not involve the computer! In other words, Harry is collecting information that is not going to go into the computer. Needless to say, Harry is able to look out the window and use similar forms of data collection. This, then, is the reason that Harry can make some decisions that cannot be made centrally, either by people or a computer.

Figure 2

The problem is that Harry is a fairly low-cost instrument for collecting information and performing not just simple planning functions, but actually quite sophisticated operational strategies, as long as they do not require information from other subproblems (Harry is no better at learning what is going on elsewhere in the network than anyone else).

The information wall

In today's information age, we often harbor the goal that we will be able to centralize all the information about our system so that people, typically with computerized assistance, will be able to make the best possible decisions. With the vast expansion in information technologies, this is a reasonable goal. But there is a fundamental tradeoff between our ability to eliminate errors in our knowledge of a system, and the cost of eliminating these errors. While we may have the technology to know almost anything we want about our problem, at some point the cost of doing so exceeds the business benefit, creating what I like to call an *information wall*, depicted in figure 3. On one side of the wall is information that is generally known to the company (for example, it is available in the computer); on the other side is Harry, doing the best he can, but doing better than anyone else because he has the information, and the rest of us don't.

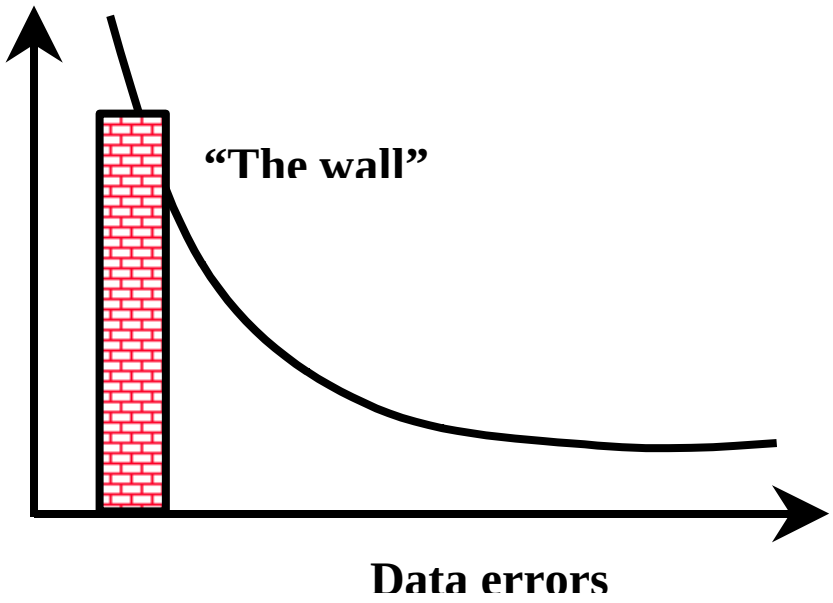

Figure 3

We all understand that when we formulate a model, we should try to capture the real problem as accurately as possible, recognizing the usually tradeoffs between precision and tractability. An overlooked dimension of problem realism is modeling how decisions are really made, and the information available to each decision maker when he or she makes a decision. I call these *informational subproblems*. Because of the presence of the information wall, we are not going to be able to get rid of "Harry." This means that any system we build must ultimately be understandable to Harry, who will be looking at the problem from the perspective of his part of the world.

Controlling complex systems

Our challenge of making decisions without complete information brings us back to the task: how do we manage complex systems? If we return to our original optimization problem, we probably formulated our problem in terms of the flows of various resources, captured by our decision vector x. Let's call these primal controls. Of course, for every primal problem there is the dual, formulated in terms of a set of prices that we might as well call p. The operations research community has for many years formulated problems using this nice dichotomy.

It is my claim, however, that the control of complex systems can be divided along three primary dimensions:

- Primal controls – Here we cover decisions that involve the traditional management of physical resources.
- Dual controls – These cover prices and other parameters that affect the value of a decision.
- Information controls – Here, we include decisions that govern what information is made available to each decision maker.

It is somewhat ironic that while the operations research community (including myself) spends much of its time trying to solve for primal controls, many managers spend most of their time with dual and, most

notably, informational controls. American industry has spent billions in the last decade on new information systems; a decision to provide information is an indirect way of controlling a system. In the presence of incomplete information, we can best help Harry not by telling him what to do, but by giving him more information so that he can make better decisions.

Formulating a subproblem

If we are going to address this issue formally, we need to model informational subproblems. If *x* is our vector of decisions and *q* is our index for a subproblem, we can let x_q be the vector of decisions for subproblem *q*, where $x_q = \{x_{dq}\}_{d \in D_q}$. Now let x_q^p be the *function* that makes these decisions, where the superscript p denotes a *policy*, and where Π is a family of policies. We might let Π^c be the set of policies in class *c* consisting of a single, functional form. Policies $p \in \Pi^c$ would represent different values of any parameters that would be used to characterize policies in a particular class.

Of course, a decision function depends on the information available to the decision maker. We let I_q denote the information set for subproblem *q*. As an illustration, assume that our class *c* is a set of linear programs, so that our decision function looks like:

$$x_q^p(I_q) = \arg\min c_q x_q$$

subject to:

$$A_q x_q = b_q$$

$$x_q \geq 0$$

In this case, we would have $I_q = \{A_q, b_q, c_q\}$. Not surprisingly, we say that "I_q" is the "IQ" of our subproblem.

Our path to making better decisions starts to become clear. To get better performance from subproblem *q*, we have to raise its "IQ." First, however, we have to understand just what we mean by information.

What is information?

To provide better information, we have to first settle on what it is, and then turn to identifying the major classes of information. Our approach to information is quite different than that used within the field of information theory (see, for example, Cover and Thomas, 1991) where the emphasis is on data representation, whereas we are interested in information only as it supports decisions. We begin by distinguishing data, knowledge and information.

For our purposes, data is anything that can be stored in a computer data file (in other words, bits and bytes). The concept of knowledge is a bit more involved. Knowledge comes in two forms: data knowledge, and functional knowledge. Data knowledge is data produced by an exogeneous source. Functional knowledge is functions (both the form and the parameters) that help us to improve our knowledge about something that is not fully known. For example, data knowledge may be the history of demands for a product or the parameters of a regression model relating demands to calendar effects and other exogenous information. The prediction of the demand next week represents a kind of information, but it is not knowledge (it was derived from the knowledge of the historical patterns and the functional relationship between past and future demands).

Information, finally, is data that helps us make a decision (somewhat more formally, information is a type of data that under at least some circumstances will change a decision). If the provision of a piece of data never has an impact on a decision, then it is not information (we might call it entertainment).

Our system evolves over time because of the arrival of information. We distinguish two primary sources: exogenous information (customer demands, weather delays, equipment failures) and decisions, which we view as a type of endogenous information. Sometimes exogenous and endogenous information sources have the same impact (exogenous: the plane broke down; endogenous: we decided to put the plane in for maintenance). Regardless of the source, they both have the effect of changing our system.

I claim there are four classes of information:

- K_q^D - Knowledge (data).
- Ω_q - Forecasts of exogenous information processes (forecasting).
- x_q^p - Forecasts of endogenous information processes (planning).
- V_{M_q} - Forecasts of the impact of a decision in one subproblem on another (we call these *values*).

Because these information classes are so fundamental, a brief discussion of each is warranted.

Data knowledge

Most planning systems focus on expanding the knowledge database K_q^D. We might say that the information age has focused on increasing this set. We can formalize the set K_q^D with just a little more notation. Let C^I be the set of information classes, and let E_q^c be the information elements for subproblem q within a class (for example, $c \in C^I$ may be the set of trucks we are managing, and $\mathsf{E}^{(truck)}$ would be the actual list of trucks). Typically, an information element $e \in \mathsf{E}^c$ would have a set of attributes, which we can represent as $a_e \in \mathsf{A}^c$. If we have multiple decision makers, we would assume

that decision maker *q* has access to the information elements in the set E_q. Now, we can represent our (data) knowledge base as:

$$K_q = \{a_e \mid e \in \mathsf{E}_q\}$$

Forecasts of exogeneous processes

Forecasts of exogenous processes, represented by Ω_q, capture basics such as demand forecasting, but can also include forecasts of people behavior (e.g. predicting the number of factory workers who will call in sick on a given day), equipment behavior (how many locomotives will be out of order on a given day), and weather.

The vast majority of forecasting systems make a single projection of future events. This is equivalent to assuming that Ω_q has a single element (the *point* forecast). This is understandable for its simplicity, but surprising because it is so unrealistic. People routinely make decisions that reflect the distribution of outcomes when they make allowances for safety stock or extra time in a schedule.

Forecasts of endogenous information processes (planning)

This category covers plans that have been made previously, which are then an input to our process. Strictly speaking, this is a form of knowledge, because it is exogenously derived information, but we prefer to group it alongside forecasts of exogenous processes. We represent a plan using the notation x_q^p. This notation is similar to our (primal) decision vector x_q, but in practice, the plan is usually made at a more aggregate level. The decision vector x_q is often very detailed (for example, which job goes on which machine at a point in time), whereas a plan is almost always aggregated up to a more meaningful level (the total number of jobs to be processed on a machine on a given day). It is useful to think of a

plan as a *forecast* of a decision. In this way, an aggregated set of decisions (a "plan") is a better forecast of a future decision since a detailed plan is unlikely to be a good predictor of actual future decisions.

It is very common to formulate optimization models without incorporating prior plans (the concept is that the model is being used to develop a plan). This is surprising, because almost no company could run this way; it is important when planning to be sure that decisions do not deviate too much from previous plans. Interestingly, there is a strong theoretical foundation for incorporating prior plans into current decisions. Proximal point algorithms, originally developed by Rockafellar (1976), use the basic form:

$$x^{n+1} = \arg\min cx + r\left\|x - x^n\right\|$$

where the decision at iteration n+1 should not be "too far" from the decision at the previous iteration, given by x^n. If a company revises its plan each month, then we would view x^n as last month's plan, and x^{n+1} would be next month's plan.

Planning is fundamental, but it is not always explicit. There are three types of "plans":

- Plans – These are explicit sets of decisions at some level of aggregation.
- Patterns – Even when a company does not use an explicit plan, people acquire behavioral patterns, generally in the form of $state \times action$ pairs. Humans acquire these patterns over time, and they help to provide stability to an operation over time. Of course, what some people view as stability others view as behavioral inertia. Even if these $state \times action$ pairs are not explicit, the tendency of humans to follow them can have as powerful an effect as any plan. Models which deviate too dramatically from historical patterns typically have a difficult time being implemented.
- Policies – These are explicit rules (typically in the form of $state \times action$ pairs) set by management that prescribe actions under certain conditions. These are similar to patterns, but they are set explicitly by management.

It is important, in the formulation and implementation of models, to understand the role of plans, patterns and policies. As much as management wants to use a model to overcome behavioral inertia, there are both good as well as bad patterns of behavior in any organization, and the challenge is sorting these out. A model can incorporate historical patterns explicitly, running the risk of repeating past mistakes, or ignore these patterns, running perhaps the even greater risk of ignoring the lessons of years of experience developed by knowledgeable operations professionals.

Value functions

Value functions represent the ability to estimate the impact of a decision made by one person (subproblem) on another part of the system. A value function captures the impact of a decision on the objective function, and hence the units are always in dollars (or the units of the objective function). The simplest illustration of a value function arises when purchasing something from an outside supplier. Assume you wish to purchase a quantity x, and the price of the item is p. The amount you will spend, then, is px. This is an instance of a linear value function.

Value functions are often ignored, but are widely used in the technical literature (see, for example, Puterman, 1994, Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 1998 for discrete dynamic programs and Sen and Higle, 1999, Birge and Louveaux, 1997 for stochastic linear programs). A value function can be derived from dynamic programming (where they are often called value functions but sometimes are referred to as cost-to-go functions), and stochastic programming (where they are called recourse functions, and are sometimes approximated using Benders cuts). In the context of multistage linear programs (which arise in many resource allocation problems), a value function can be viewed as a forecast of a dual variable.

Summary

We have, then, four different types of information which we represent, notationally, by K_q, Ω_q, x^p and V_{M_q}. Using these types of information, we can construct different decision functions. Not surprisingly, the design of the information set and the design of the information function need to go together. For this reason, just as we used the index p to indicate the type of decision function in x_q^p, we can use the same notation to capture the type of information set, as in I_q^p. Four useful classes of information sets include

$$\Pi^M = K_q$$
= Myopic information set (based purely on data knowledge)

$$\Pi^{RH} = \left\{K_q, \Omega_q\right\}$$
= Rolling horizon procedures (which combine knowledge with forecasts)

$$\Pi^P = \left\{K_q, \Omega_q, x_q^p\right\}$$
= Knowledge combined with forecasts of exogenous and endogenous information processes.

$$\Pi^V = \left\{K_q, V_{M_q}\right\}$$
= Knowledge and value functions.

These four classes represent the most popular ones in practice (the first two) or in the technical literature (the last two). Of course, it is possible to create a fifth class that we might call "all of the above", but I have never seen this implemented either in practice or in the academic literature.

The class Π^M covers myopic policies that use only knowledge, without any attempt to forecast the future. Most people who refer to "policies" are referring to myopic policies. Procedures that use forecasts of exogenous events, Π^{RH}, are typically called rolling horizon procedures, and the vast majority of these use deterministic forecasts ($\left|\Omega_q\right| = 1$). Decision functions in the class Π^P incorporate

previously designed plans, patterns or policies; conveniently, a standard method for incorporating these is via a proximal point algorithm, a convenient mnemonic device as well as a powerful planning tool. These algorithms would use plans (or patterns or policies) as follows:

$$x^{n+1} = \arg\min cx + r \left\| G(x) - x^P \right\| \quad (5)$$

where $G(x)$ is a function that aggregates the decision vector x up to the same level as the plan x^p. Value policies, represented by Π^V, are perhaps the most difficult class of policies to develop and implement, but these are also the only class that have a theoretical basis for optimality for general problems.

Each of these classes of information creates a different class of decision function. Which one is best? Obviously, that depends on the class of problem, but I would rank the four classes in terms of sophistication in the same order they are listed above. Myopic policies are the easiest to implement and can be optimal in very specialized circumstances, but rarely in practice. However, for many companies, simply increasing the knowledge set K_q can be an important step. The transition from a myopic information set to a rolling horizon procedure which incorporates forecasting is typically the second step that companies take to improve decision making (after increasing the knowledge base). Incorporating plans or value functions represent successively higher levels of sophistication.

Controlling the flow of information

The important message, of course, is that we control behavior by controlling information. More precisely, we can control who has access to what information. We start by assuming that every information element *e* exists in a set E_q for some *q* (the same element may be in more than one set).

The information challenge arises when q is a human, and E_q represents head knowledge (for example, the retail distributor who is the only one who sees the status of shelf inventory, or the yard supervisor at a railroad who is the only one who knows that the locomotive is out of order). Assume now that we wish to send a piece of information in E_q to another set $\mathsf{E}_{q'}$ (q might be a person, while q' might be a different person, a local computer, or a central database). We might let $q_{e,q,q'} = 1$ if we decide to send element $e \in \mathsf{E}_q$ over to set $\mathsf{E}_{q'}$, and let q_q be the vector of all information exchange decisions for subproblem q. The parallels between our primal decisions x_q and our information decisions q_q are quite close. We might, for example, let $c_q^X(x_q)$ be the cost of moving physical resources, and $c_q^I(q_q)$ be the cost of moving informational resources. If I_q is the information available to subproblem q, then we would write $I_q(q)$ to capture the dependence of information on our decisions to flow information. Furthermore, we note that our decisions $x_q = x_q^p(I_q) = x_q^p\left(I_q(q)\right)$ are also, indirectly, a function of our informational decisions. This means that we can now write an objective function for flowing information that reflects both the cost of moving information, and the impact of these decisions on the flows of physical resources:

$$\min_q \sum_q C_q^X\left(x_q^p\left(I_q(q)\right)\right) + C^I(q) \tag{6}$$

Of course, the cost function for information flow is going to be similar to the types of cost functions that we often encounter for flowing physical resources.

To complete our optimization problem, we have to fill out a set of constraints. When we formulate models of physical resources, we typically include three types of constraints: flow conservation, upper bounds reflecting restrictions on the rate of flow, and nonnegativity or integrality constraints. When we

turn to formulating constraints on the flow of information, we find that one notable constraint is missing… flow conservation! With information, I can share information with others (without giving up the information myself). What is particularly interesting is that I can give away information that I do not even have! (This is called lying). The handling of false or unreliable information is a skill that humans manage particularly well, and which computers manage particularly badly.

The cost function $c_q^I(q_q)$, as with the more traditional cost function for the flow of physical resources $c_q^X(x_q)$, may take a variety of functional forms. Linear functions might be used to describe telephone calls, reading the newspaper, or sending emails, faxes and letters. By adding information technology (for example, developing a new database) we may lower the marginal cost of flowing information (the phone call is replaced with the cost of bringing up a computer screen) but we incur the fixed charge of developing and implementing the database. A convex cost function would be used to describe the increasing cost of trying to communicate too much information (think about filling out a long questionnaire or reading a long email), while a concave cost function would describe the lower marginal cost of adding information to an exchange (it is easier to send and receive a three-line email than three one-line emails).

The flow of information, controlled by the vector q_q, can describe the movement of any of our four types of information. Most of the time, q_q is describing the movement of knowledge from one source to another. These flows have the effect of increasing K_q. However, q_q can be used to capture the decision to implement a new forecasting system (thereby adding Ω_q to the information set I_q), a new planning system (which adds a plan x_q^p to the information set, and implies the use of a decision function similar to (5)), or the use of value functions V_{M_q}.

Conclusions

The thoughts in this paper have grown out of a two-decade long career of implementing computer models for complex operations arising in transportation and logistics. The successful models can be easily separated from the unsuccessful ones based on an understanding of human decision functions and information. Planning models are often the most successful because both computers and humans tend to have access to the same information. As models get closer to real-time operations, we encounter the information wall, which means we are ultimately dependent on the human making the decision with the computer playing a supporting role.

In this setting, we have to carefully understand the human's decision function ($x_q^p(I_q)$), and to make sure that we are providing *information* (data that may impact a decision) and not entertainment. Here, we encounter another wall. Human decision making fundamentally follows the paradigm of *state*×*action* pairs (when in this state, use this action), which is the foundation of much of artificial intelligence. By contrast, computer models depend on costs and value functions (in any of its major disguises). Providing humans with this same information is of relatively little value. Instead, we depend on computer models to develop *plans*, which humans have an easier time of following. However, it is important that computer models communicate these plans at an appropriate level of aggregation. Just as an overly detailed forecasting model is of little value, an overly detailed decision function (which can be viewed as a kind of forecasting function) should make decisions at an appropriate level of aggregation.

Traditional computer models focus on planning the flows of resources. In this paper, the focus has been on modeling the flow of information, and building decision functions that capture the kind of information that will actually be available to a real decision maker when a decision is actually made.

When we make this step, we can then focus on understanding the value of information, and determining which information is cost effective to provide.

References

Bertsekas, D.P. and J.N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, Massachusetts, 1996.

Birge, J.R. and F. Louveaux, *Introduction to Stochastic Programming*, Springer-Verlag, New York, 1997.

Cover, T.M. and J. A. Thomas, *Elements of Information Theory*, John Wiley, New York, 1991.

Powell, W.B. and J. Shapiro, "A Representational Paradigm for Dynamic Resource Transformation Problems," Annals of Operations Research on Modeling, (Fourer, Coullard, eds.), To appear.

Putterman, M. L., Markov Decision Processes, John Wiley and Sons, Inc., New York, 1994.

Rockafellar, R.T., "Augmented Lagrangians and Applications of the Proximal Point Algorithm in Convex Programming," Math. Of Operations Research, Vol. 1, pp. 97-116 (1976).

Sen, S. and J.L. Higle, "An Introductory Tutorial on Stochastic Linear Programming Models," Interfaces, Vol. 29, No. 2, pp. 33-61 (1991).

Sutton, R.S. and A.G. Barto, *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts, 1998.