CHAPTER 13

LEARNING IN MATHEMATICAL PROGRAMMING

There are applications where we have to collect information to be used in a larger optimization problem such as a shortest path problem or linear program. Ranking and selection can be viewed as the solution to a simple linear program

$$v^n = \max_x \sum_{i=1}^{M} \mu_i x_i$$

subject to

$$\begin{aligned} \sum_{i=1}^{M} x_i &= 1 \\ x_i &\geq 0. \end{aligned}$$

The solution to this linear program requires sorting (μ_i) for all i and choosing the index $i*$ with the largest value of μ_i, which gives us a solution where $x_{i*} = 1$, and $x_i = 0,\ i \neq i*$. Our learning challenge is to choose an element j to measure, producing a value $v^{n+1}(j)$ computed using the vector μ_i. The goal is to choose the index j that maximizes the expected value of v^{n+1}. Note for this discussion that we are using indices i and j for our choices, and we are switching (for this chapter) to x as the vector of implementation decisions.

Optimal Learning. By Warren B. Powell and Ilya O. Ryzhov

We can take this perspective one step further. Assume now that we have a more general linear program which we write as

$$\min_x \sum_{i \in \mathcal{I}} c_i^T x_i$$

subject to

$$\begin{aligned} Ax &= b, \\ x_i &\geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

Here, x is a vector with potentially hundreds or even tens of thousands of dimensions. More significantly, the A matrix might be general, although there are specific problem structures that are of interest to us.

Linear programming has numerous applications, including a broad class of problems on networks, where the matrix A is used to represent flow conservation equations. The objective can be to minimize the travel distance across the graph (shortest-path problems), or to efficiently move resources between nodes (such as the "transportation problem" or general network flow problems). Many other optimization problems can also be expressed as linear programs. A classic example is the problem of formulating a production plan to maximize profit. A company produces M products, and wishes to produce x_i units of product $i = 1, ..., M$ to maximize the total profit $c^T x$ with c_i being the profit from selling one unit of product i. The products are made using J different resources, with b_j being the total amount of resource j available. The matrix A denotes the production constraints, with $A_{j,i}$ being the amount of resource j needed to create one unit of product i.

In the basic LP model, we assume that the parameters A, b and c are known. In reality, we are unlikely to know them exactly, just as we do not know μ in ranking and selection. Suppose that our company has developed a new product, and now needs to decide how much of it should be produced. We may have an estimate of c_i based on some preliminary data. Perhaps our sales figures for our old line of MP3 players may give us some idea of the profitability of the new line. At the same time, we are still quite uncertain about the true value of c_i. But now, suppose that we have the ability to collect additional information about c_i before we commit to a production plan x. Perhaps we have a chance to run a test market where we can get a sense of how well product i might perform. The results might change our estimate of c_i, thus affecting our final production plan. Given our limited time and money, which products should be included in the test market? Or, in other words, which information will help us make the best possible decision?

Optimal learning has a role to play here, but it is not a simple matter of picking up and applying the formulas and algorithms from Chapters 3 and 4. In mathematical programming applications, our *experimental decision* (say, measuring the coefficient c_i) is distinct from our *implementation decision*, which we now represent as the vector x. We may learn about a single product or a single region in the network, but our overall goal is to solve an optimization problem. The coefficients $i = 1, ..., M$ in our LP are a bit like the "alternatives" from Chapter 3, in that we have a choice of which coefficients to learn about, but we are not simply interested in finding the largest coefficient. The part should teach us about the whole; we should learn about

those coefficients that contribute the most to our ability to optimize. Running a test market for product i should lead us to a better production plan.

Although we cannot recycle the knowledge gradient formulas from Chapter 4 in a straightforward way, we can still apply the concept of the knowledge gradient. In this chapter, we show how the value of information can be expressed in problems with uncertain objective functions. After describing several applications, we illustrate the application of the knowledge gradient first in the context of simple shortest path problems, and then for more general linear programs.

13.1 APPLICATIONS

In this section, we introduce three models where the implementation decision requires solving a mathematical program. We begin with a model for piloting a hot air balloon, where the challenge is moving the balloon up and down to find the fastest wind. Given a current set of estimates about the wind, the balloon can solve a linear program to find the best path to the destination. Then, we describe a common problem in the investment community where it is necessary to visit different companies to learn about their potential for future profits. The challenge is determining which companies to visit (we do not have the time to visit all of them), after which we have to take estimates of expected returns, variances and covariances to determine the best portfolio. We close with a discussion of graph problems.

13.1.1 Piloting a hot air balloon

Consider the problem faced by a balloonist trying to get from one point to another in the presence of changing winds that can be different at different altitudes. Her ability to make progress depends on finding wind that moves in the right direction. She observes the speed and direction of the wind at her current location, but she may wonder if the conditions are better at a different altitude. Observing the wind at one altitude provides some information about the wind at other altitudes, as well as the wind at later points in time. But it takes time and (if she has to raise her altitude) energy to make these observations.

The model Optimizing the trajectory of a hot air balloon is a fairly difficult control problem, even when we do not include learning as a dimension. For our purposes, we are going to simplify the problem by assuming that our balloon can only move in two dimensions: forward and up or down. Our interest is in finding a path that balances distance against changing (and uncertain) wind speeds so that we get to the destination as quickly as possible.

Our view of the problem is depicted in Figure 13.1, where our balloon is currently at horizontal location x, vertical location y, and is faced with one of three decisions: d^{up} takes the balloon on an upward trajectory, d^{stay} holds the balloon at its current altitude while d^{down} takes the balloon on a downward trajectory. Although we would like to model energy expenditure, we are only going to capture the time required

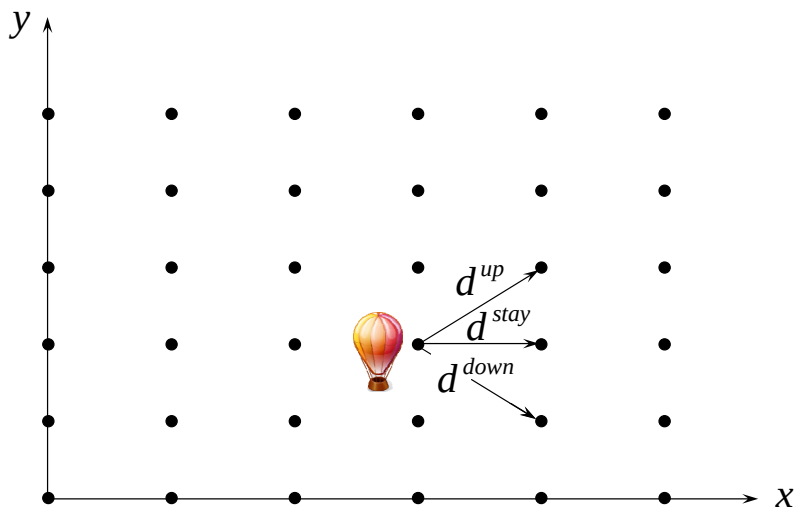


Figure 13.1 Configuration of a two-dimensional hot air balloon trajectory.

to complete each decision. We assume that the balloon moves 30 percent slower moving up, and 20 percent faster moving down.

The physical state $R^n = (x^n, y^n)$ captures the location of the balloon. Our belief state about the wind is given by $B^n = (\theta^n_{xy}, \sigma^{2,n}_{xy})_{x,y}$ where θ^n_{xy} is what we think the wind speed is at time n, location x, y, and $\sigma^{2,n}_{xy}$ is the variance in our distribution of belief. When the balloon is at location (x, y) at time n, the true wind speed is $\bar{\mu}^n_{xy}$ which we measure with noise according to

$$W^n_{xy} = \bar{\mu}^n_{xy} + \varepsilon^n.$$

We assume experiments are independent across space and time, and are normally distributed with mean 0 and variance λ^ϵ. The true wind speed also evolves over time according to

$$\theta^{n+1}_{xy} = \bar{\mu}^n_{xy} + \hat{\mu}^{n+1}_{xy},$$

where $\hat{\mu}^{n+1}_{xy}$ describes the actual change in the wind from time period to time period. The random changes in the wind, $\hat{\mu}^n_{xy}$, are correlated across both space and time, since if the wind is high at time n, location (x, y), then it is likely to be high at nearby locations, and at the same location at later points in time. The covariance between wind observations at two different altitudes at the same point in time is given by

$$Cov(W^n_{xy}, W^n_{xy'}) = \sigma^2_W e^{-\beta_y |y-y'|} + \lambda_{xy}.$$

Similarly, we assume the covariance between a location x and a location $x' > x$ at the same altitude is given by

$$Cov(W^n_{xy}, W^n_{xy'}) = \sigma^2_W e^{-\beta_x |x-x'|} + \lambda_{xy}.$$

We combine these two to give us a general covariance structure

$$Cov(W^n_{xy}, W^n_{xy'}) = \sigma^2_W e^{-\beta_y|y-y'|-\beta_x|x-x'|} + \lambda_{xy}.$$

Using this function, we can construct a covariance matrix Σ^n with elements $\Sigma^n_{xy,x'y'}$. We use this structure to initialize our covariance matrix Σ_0. Next let e_{xy} be a vector of 0's with a 1 corresponding to the location (x, y). The general formulas for updating the vector of estimates of the mean and covariance matrix for our distribution of belief about the wind is given by

$$\begin{aligned}
\theta^{n+1} &= \Sigma^{n+1}\left((\Sigma^n)^{-1}\theta^n + (\lambda_{xy})^{-1}W^{n+1}e_{xy}\right), \\
\Sigma^{B,n+1} &= ((\Sigma^n)^{-1} + (\lambda_{xy})^{-1}e_{xy}(e_{xy})')^{-1}.
\end{aligned}$$

This gives us the updated estimate of the mean and variance, where the variance takes into account the experimental error. But this ignores the changing velocity due to $\hat{\mu}^n$. When we take this into account, our updated covariance matrix is given by

$$\Sigma^{n+1} = \Sigma^{B,n+1} + \Sigma^{\mu}.$$

So, $\Sigma^{B,n+1}$ will shrink relative to Σ^n, but it will grow due to the effect of Σ^{μ}. We note that this is not a very realistic model for wind. For example, if we did not run any experiments (and possibly even if we do), the covariance matrix will grow without bound. However, this model will illustrate an interesting learning problem without becoming buried in the details of the physics.

Planning a trip The next step is to plan a trip given what we know now. We use our current estimates of speeds at each altitude. Without additional information, our best estimate of the speed in the future is the same as it is now, for each altitude, which means that $\theta^n_{xy} = \theta^{n'}_{x'y}$ for $n' \geq n$ and $x' \geq x$. Using the assumption that speeds are 30 percent slower going up and 20 percent faster going down, we can construct a speed graph over the entire distance, similar to what is depicted in Figure 13.2. If we convert speeds to travel times, we now have a simple shortest path problem to find the fastest path from the current location to the end (presumably at an altitude of 0).

This problem can be formulated as a linear program by defining the decision variable

$$u_{xy,x'y'} = \begin{cases} 1 & \text{if we decide to make the move from } (x,y) \text{ to } (x',y') \\ 0 & \text{otherwise.} \end{cases}$$

This entire vector is chosen at time n, and can be thought of as the current planned trajectory given what we know now. From a particular point (x, y), we assume that there are only three points that can be reached, by using the decisions up, down or stay. Let $c_{xy,x'y'}(\bar{\mu}^n)$ be the time required to move from (x, y) to (x', y'). Our linear program would be given by

$$F^n(\bar{\mu}^n) = \min_u \sum_{x,y} \sum_{x',y'} c_{xy,x'y'}(\bar{\mu}^n) u_{xy,x'y'} \tag{13.1}$$

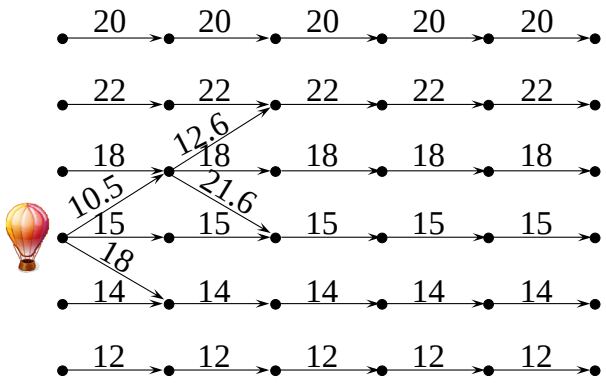


Figure 13.2 Speeds for each decision at each (x, y) location given the current belief state.

subject to

$$\sum_{x'y'} u_{xy,x'y'} = R_{0y}, \tag{13.2}$$

$$\sum_{x'y'} u_{xy,x'y'} - \sum_{xy} u_{xy,x'y'} = 0, \tag{13.3}$$

$$u_{xy,x'y'} \geq 0. \tag{13.4}$$

Here, $R_{0y} = 1$ if the balloon starts the process at an altitude y. Equation (13.3) requires that if the balloon arrives to (x, y) then it must also leave. Equation (13.4) requires that the flows be nonnegative. We would also like the decision u_t to be 0 or 1, but we will get this as a natural result of the structure of the problem.

We are assuming that we are solving the problem deterministically, using our current estimates of the speeds. We could put probability distributions around the speeds and solve a stochastic optimization problem, but this would not contribute to our understanding of how to handle the learning aspects of the problem.

Learning while flying Imagine that the wind at your altitude has dropped significantly. You are, of course, wondering if the wind is better at a different altitude. If we move to (x', y'), we are going to observe $W^{n+1}_{x'y'}$. Because of the covariance structure, this observation will allow us to update our estimate of the entire vector θ^n giving us $\theta^{n+1}(W^{n+1}_{x'y'})$. This would allow us to solve the problem $F^{n+1}(\theta^{n+1}(W^{n+1}_{x'y'}))$ using our new belief state. We have three possible decisions (up, down or stay). If we want to use pure exploitation, we would solve the problem $F^n(\theta^n)$, which would tell us which action to take. But this ignores the potential value of learning. As an alternative, we could choose the best value of $(x'y')$ (corresponding to one of the three decisions) to solve

$$\max_{(x'y')} \mathbb{E}^n \left\{ F^{n+1}(\theta^{n+1}(W^{n+1}_{x'y'})) - F^n(\theta^n) \right\}. \tag{13.5}$$

We have written this function in the form of the knowledge gradient, recognizing that $F^n(\theta^n)$ is a constant (given what we know at time n) which does not affect the solution.

When we first introduced the knowledge gradient, we presented it in the context of ranking and selection where we had to choose from among a small, discrete set of choices. Now, we have to solve a linear program for each possible experiment, which means we can no longer compute the expectation exactly. But we can take advantage of the fact that we have a small number of potential experimental decisions (up, down or stay), which means we effectively face a ranking and selection problem. We cannot compute the expectation exactly, but we can use Monte Carlo methods. Let $W^{n+1}_{x'y'}(\omega)$ be a sample observation of the wind if we were to choose to go to location $(x'y')$. This is not a true observation - it is a simulated observation for the purpose of approximating the expectation.

Given this observation, we can update the mean and covariance matrix and solve the linear program. Note that this observation does not just change the speed out of node $(x'y')$ (take another look at Figure 13.2); it changes estimates of speeds over the entire network, partly because of the presence of correlations, and partly because we solve our linear program by assuming that as we step forward in time, we do not collect additional information (which means that speeds at the same altitude for larger values of x' are the same). Let $F^{n+1}(x'y'|\omega)$ be the resulting "sample observation" of $\mathbb{E}^n F^{n+1}(\theta^{n+1}(W^{n+1}_{x'y'}))$. We could then repeat this, say, 100 times, for each possible decision and take an average. We could then make the decision that has the highest estimated value.

Sound familiar? This is precisely the ranking and selection problem. Solving the linear program 100 times and taking an average is a bit clumsy (famously known as the "brute force" solution). A more elegant approach would be to use (drum roll please) the knowledge gradient algorithm to choose the best decision. This would allow us to avoid running 100 experiments of a choice that does not look promising. We can build a prior by initially evaluating each choice assuming that we do not learn anything (the wind does not change from the prior), so that we start with a solution built around the pure exploitation policy.

Optimal learning versus stochastic optimization It is useful to contrast our learning algorithm versus what we might have done if we were solving this as a stochastic optimization problem. In a stochastic optimization model, we would acknowledge that we do not know what wind we will face when we arrive to a location $(x'y')$ (above, we solved the problem deterministically). For example, a deterministic solution might take us closer to the ground if the wind seems to be fastest there. However, a stochastic solution might recognize that the wind near the ground might be much lower than expected, and if we are close to the ground, the only place to go is up (at a high cost). A stochastic solution to this problem is relatively difficult, and probably requires the techniques of approximate dynamic programming.

The issue of whether to use a deterministic or stochastic algorithmic strategy did not arise with the simpler problems such as ranking and selection, because the optimal solution was to pick the index $j*$ with the highest expected value of θ^N_j. Our stochastic shortest path problem, however, is different because it involves finding a

path over a sequence of steps with uncertain costs. Even if we fix our distribution of belief, it is still a stochastic optimization problem.

The essential difference between a stochastic optimization algorithm and a learning algorithm is that with a learning algorithm, we accept that observations of the wind (the external experiment) come from an exogenous distribution that may be different from our true distribution of belief. As a result, we use these observations to update our distribution of belief. With a classical stochastic optimization formulation, we acknowledge that we do not know the actual wind, but we do know the probability distribution, which means that observations do not change our distribution of belief about the distribution.

13.1.2 Optimizing a portfolio

Imagine that you are looking at investing in a group of companies. The companies are organized by industry segment (retailing, financial services, transportation), so we will refer to the j^{th} company within the i^{th} industry segment. We start by assuming that θ^0_{ij} is the expected return (over some period of time) for company j in segment i. We also let Σ^0 be the matrix that captures the covariances in our beliefs about μ_{ij}, where the diagonal elements $\Sigma_{ij,ij}$ represent the variance.

In general, we assume that the covariance between two companies has the structure

$$\Sigma_{ij,i'j'} = \sigma_0^2 + \sigma_i^2 1_{\{i=i'\}}.$$

Thus, σ_0^2 captures the common covariance (e.g. the extent to which companies respond to the general economy), and σ_i^2 captures the common covariance for companies within industry segment i.

Before we invest in a company, we have the ability to visit the company and learn more about the management team, marketing plans and facilities. We might assume that our observation W of each company is independent with variance λ_{ij}. Let $x_{ij} = 1$ if we visit company (i, j). If we choose to visit company (i, j), we can update the mean and variance using our standard formulas, giving us an updated $\theta^1_{ij}(x)$. If $x_{ij} = 0$, then $\theta^1_{ij}(x) = \theta^0_{ij}$, and we let $\Sigma^1(x)$ be the updated covariance matrix.

Given the vector x of visits (presumably chosen to a constraint on how many visits we can make), we then have to find an optimal portfolio. Let y_{ij} be the amount that we are going to invest in company (ij). We do this by solving the standard quadratic programming problem

$$F(x) = \mathbb{E}F(x, W) = \min_y \left(\theta^1(x)y + \theta y^T \Sigma^1(x) y\right) \tag{13.6}$$

subject to

$$\sum_{ij} y_{ij} = R^I, \tag{13.7}$$

$$y_{ij} \geq 0. \tag{13.8}$$

Here, x is our experimental decision, W captures what we observe if we choose to run the experiment, and y is our implementation decision. R^I is the amount of money

we have to invest. The learning problem is given by

$$\min_x \mathbb{E}F(x, W) \tag{13.9}$$

subject to

$$\sum_{ij} c_{ij}x_{ij} = R^M, \tag{13.10}$$

$$x_{ij} \geq 0 \text{ and integer.} \tag{13.11}$$

Here, R^M is our experimental budget. The learning problem is hard to solve, both because it requires integer solutions but primarily because of the problems computing the expectation in (13.9). We suggest simply finding the marginal value of making each visit, and then assigning visits in order of decreasing marginal value until the budget is exhausted. This is a type of batch ranking and selection problem, where instead of finding the best choice, we find the best set of choices.

Even this simple heuristic is hard to implement. Imagine that there are 100 companies that we are considering, and we have the budget to make 20 visits. What are the best 20 companies to visit? We are not able to compute the expectation $\mathbb{E}F(x, W)$ in (13.6) either. Instead, we can choose a company to visit, then randomly sample what we might learn $(W_{ij}(\omega))$. We can perform 100 samples of each company and take an average to obtain an estimate of the value of visiting each company, but this may be computationally demanding (and 100 samples may not be enough). Or, we can use the knowledge gradient algorithm to allocate our computational budget across the 100 companies.

13.1.3 Network problems

Optimization on a graph is a well-known class of mathematical programming problems, with applications in transportation, project management, telecommunication and other areas. A graph is described by a set $\mathcal{V}$ of nodes (or "vertices") and a set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of edges, where each edge connects two nodes. Figure 13.3 provides a visual illustration. It is easiest to think of a graph as a map of physical locations: for example, nodes could represent cities and edges could stand for major highways connecting those cities. An edge $(i, j) \in \mathcal{E}$ between nodes i and j carries a cost (or "length") c_{ij}. The cost can be expressed in terms of money or time. For example, we might focus on the economic cost of transporting a shipment of goods from one city to another, or on the time needed for the shipment to reach its destination.

Implementation decisions in networks The shipment problem is a well-studied application of *minimum-cost flows*, a model for optimizing the route of resources across a network. A number b_i represents either the supply or demand for a commodity at node i. If $b_i > 0$, then i is a supply node (perhaps we have a production facility in city i, or we purchase from a manufacturer located in this city). If $b_i < 0$, then i is a demand node (perhaps retailers in this city are selling our products). If $b_i = 0$, the node is a transit node and our product can be transported through it en route to a demand node.

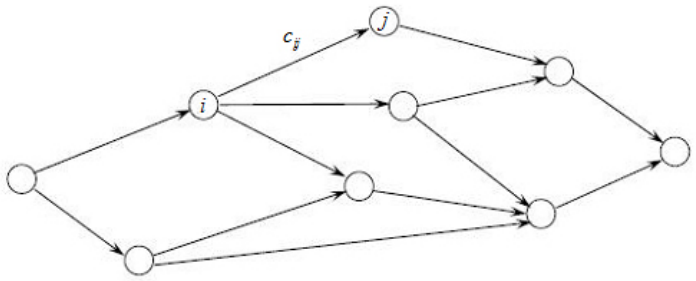


Figure 13.3 A generic network, with nodes i and j connected by an edge of length c_{ij}.

The cost of transporting a single unit of product from node i to node j is given by c_{ij}. The objective function can be written as

$$\min_x \sum_{(i,j)\in\mathcal{E}} c_{ij}x_{ij},$$

where x_{ij} represents the quantity of product that we wish to ship from i to j. Obviously we require $x_{ij} \geq 0$ for all i, j.

Each node is subject to a flow conservation constraint. Product can only enter the network through a supply node and leave through a demand node. Thus, the total product leaving node j has to be equal to the amount that entered node j, plus the amount that was supplied (or, minus the amount that was used to satisfy demand). We can write this constraint as

$$b_j + \sum_{i:(i,j)\in\mathcal{E}} x_{ij} = \sum_{i:(j,i)\in\mathcal{E}} x_{ji}, \qquad j \in \mathcal{V}.$$

The implementation decision in this problem is the flow schedule, the values of x_{ij} that we choose for all edges (i, j). The objective function and constraints are both linear in x_{ij}, making this a classic application of linear programming. There are many variations of the problem. For example, we might add capacity constraints $x_{ij} \leq d_{ij}$ if there is a cap on the amount that we can ship along a particular route (perhaps only a limited number of trucks is available). We may also add more flexibility to our model by giving ourselves the option to use only part of the available supply, in a situation where we have a choice of suppliers. In this case, for $b_j > 0$, we can add new decision variables $0 \leq y_j \leq b_j$ denoting the amount of supply we used, and replacing b_j with y_j in the corresponding flow constraints. We would also incorporate additional purchase costs into our objective function.

In some applications, we may not always know the costs c_{ij} exactly. In a supply chain, costs may depend on numerous factors. They are affected by purchase and production costs, but also by more vague factors like the reliability of a supplier. Perhaps a particular supplier has a high chance of experiencing shortages, or a particular transit route is more likely to encounter delays. Eventually, the shortages

and delays will be passed down to us, increasing our own costs. The distribution of the costs, or even their expected value, may be difficult to quantify, and has to be learned by doing business with a particular supplier or shipping along a particular transit route. We will need to model our beliefs about the costs and think about which suppliers we would like to experiment with in order to arrive at the best shipment schedule.

A special case of minimum-cost flows is the well-known shortest path problem. In this setting, there is a single supplier s with $b_s = 1$, and a single demand node t with $b_t = -1$. All other nodes are transit nodes, and the cost c_{ij} represents the "length" of edge (i, j). Our goal is to find a route from s to t with the shortest possible length. Like all the other network problems discussed here, the shortest-path problem can be formulated and solved as a linear program. There are also many optimization algorithms that are specifically tailored to shortest-path problems. We give one well-known example, the Bellman-Ford algorithm. Given a cost vector c and specified source and destination nodes s, t, the algorithm calculates the distance $V(i; c)$ from any node i to t in the following way:

1) Let $V(s; c) = 0$. For all other nodes $i \neq s$, let $V(i; c) = \infty$.
2) For each edge $(i, j) \in \mathcal{E}$, if $V(i; c) + c_{ij} < V(j; c)$, let $V(j; c) = V(i; c) + c_{ij}$.
3) Repeat step 2) a total of $|\mathcal{V}| - 1$ times.
4) If $V(i; c) + c_{ij} < V(j; c)$ for any $(i, j) \in \mathcal{E}$, the graph contains a negative-cost cycle and the shortest path length is $-\infty$ (corresponding to an unbounded LP). Otherwise $V(t; c)$ gives the length of the shortest path from s to t.

Optimal learning becomes an issue when the lengths c_{ij} are unknown. Length is often interpreted as travel time. For example, a GPS navigation system finds the quickest way to get from a specified origin to a specified destination. However, in reality, travel times are highly variable. GPS relies on certain estimates of travel time, but they may be outdated or inaccurate when a particular query is received. However, in the limited time available to make a decision, we may have an opportunity to collect a very small amount of information to help us choose a travel route. Some drivers located in different regions of the traffic network may be signed up to participate in a program where their smartphone can send information on traffic congestion (using the ability of the phone to tell how fast it is moving) to the GPS. We can choose to query a small number of smartphones, receive real-time observations of traffic congestion, and use them to update our estimates and improve our implementation decision (the route we choose).

Other network representations The graph does not have to represent a physical structure. Graphs are used in project management to represent precedence relations between different tasks or assignments. Consider the problem of managing a software project that consists of a series of tasks. Some tasks must be completed in a specific order, whereas others can be tackled in parallel. Figure 13.4 displays precedence relationships for the following set of tasks:

Each node in the graph represents a task, whereas an edge between tasks i and j indicates that i must be completed before j. In addition, each task has a duration d_i.

A	Build preliminary demo	F	Compatibility testing/QA
B	Alpha testing, user feedback	G	Train support staff
C	Preliminary market study	H	Final marketing campaign
D	Finalize design	I	Begin production
E	Plan production/distribution	J	Ship to retailers and launch

Given T tasks, we can schedule the tasks in the following way. Let x_i be the starting time of task i and solve

$$\min_x x_T$$

subject to

$$\begin{aligned} t_j &\geq t_i + d_i \qquad \text{for all } (i,j) \in \mathcal{E} \\ t_i &\geq 0 \qquad \text{for } i = 1, ..., T. \end{aligned}$$

We minimize the starting time of the last job. Another problem that often arises in project scheduling is known as the critical path problem. This is essentially the same as finding the longest path (the path with the largest total duration) from the first to the last job. The critical path is of interest to managers because it shows which jobs can be delayed without impacting key deadlines.

The duration of a particular task is rarely known exactly. Preliminary estimates are often inaccurate, leading to unplanned delays. Before beginning the project, we may wish to improve our estimates by going into our archives and analyzing data from earlier projects that involved similar tasks. The data may be cumbersome to access (many such archives have yet to be digitized), so our time is limited. It then becomes important to use our learning budget effectively by singling out the tasks that seem most important, and focusing on creating good estimates of their duration.

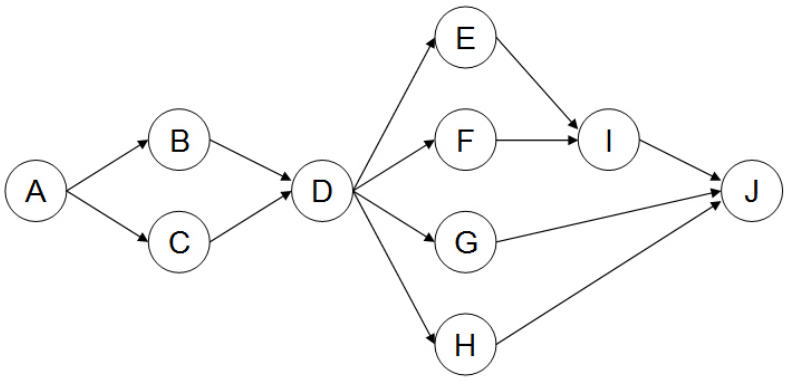


Figure 13.4 Graph of precedence relations for tasks in software development.

13.1.4 Discussion

We have just seen several problems where the decision problem (we sometimes call this the implementation problem) is a linear or nonlinear program. Finding the expected value of an experimental outcome can be computationally difficult. But we have seen that we can apply our optimal learning ideas within a learning algorithm to help us find a good learning policy.

Both of these problems could be formulated in the general framework of finding an experimental decision x to minimize $\mathbb{E}F(x, W)$. This is a very classical optimization problem with a rich algorithmic history. There are many algorithms which depend on the structure of the problem. An excellent reference for this problem class is Spall (2003) and the references cited there (the literature is extremely large).

13.2 LEARNING ON GRAPHS

We show how optimal learning can be incorporated into network optimization using, the shortest-path problem from Section 13.1.3 as an example. To make our analysis cleaner, we assume that the graph has no cycles, as in Figure 13.3.

Optimal learning begins to play a role when we no longer know the cost vector c. Let us apply our standard modeling assumptions from Chapter 4. Suppose that, for each edge $(i, j) \in \mathcal{E}$, we have $c_{ij} \sim \mathcal{N}\left(\theta^0_{ij}, \beta^0_{ij}\right)$, where β^0 denotes precision. We put a Gaussian prior on the length of each edge, and assume that the edges are independent. This is a strong assumption: for example, if the graph represents a traffic network, as in some of the examples discussed in Section 13.1.3, one would expect the travel times on neighbouring streets to be heavily correlated. We will allow correlations in Section 13.4. For now, we focus on the insights that can be obtained if we make the independence assumption.

As in ranking and selection, the belief state in this problem can be written as $S = (\theta, \beta)$, a vector of means and variances. A set of beliefs about the edges induces a belief about the best path. Define $V^n = V\left(s; \theta^n\right)$ to be the estimated length of the shortest path, given our time-n beliefs about the edge lengths. In other words, we simply run the Bellman-Ford algorithm using the estimates θ^n as the costs, and use the quantity V^n as our time-n estimate of the shortest path. Bellman-Ford can also provide us with the path that achieves this estimated length. We refer to this path as p^n.

Now suppose that, before we commit to a path through the network, we have N opportunities to collect information about individual edges. If we choose to learn about (i^n, j^n) at time n, we collect an observation $W^{n+1} \sim \mathcal{N}\left(c_{ij}, \beta^W_{ij}\right)$ and our beliefs evolve according to the familiar updating equations

$$\theta^{n+1}_{ij} = \begin{cases} \frac{\beta^n_{ij}\theta^n_{ij} + \beta^W_{ij} W^{n+1}_{ij}}{\beta^n_{ij} + \beta^W_{ij}} & \text{if } (i,j) = (i^n, j^n) \\ \theta^n_{ij} & \text{otherwise,} \end{cases} \tag{13.12}$$

$$\beta^{n+1}_x = \begin{cases} \beta^n_{ij} + \beta^W_{ij} & \text{if } (i,j) = (i^n, j^n) \\ \beta^n_{ij} & \text{otherwise.} \end{cases} \tag{13.13}$$

Just as in ranking and selection, we change our beliefs about one edge at a time. However, a small change in our beliefs about a single *edge* can have a profound effect on our beliefs about the shortest *path*. Figure 13.5(a) shows the graph from Figure 13.3, with the addition of prior beliefs about the edge lengths, represented by bell curves. The solid line in Figure 13.5(a) represents the path from node 1 to node 9 that we currently believe is the shortest. When we measure the edge $(6, 8)$, the outcome of our observation is smaller than expected, leading us to shift that particular bell curve to the left. Because this edge now seems shorter than before, we prefer to route our path through it, leading to the new solution in Figure 13.5(b). We only changed our beliefs about one edge, but the shortest path now looks very different from before.

Our objective is to choose a policy π for allocating the N experiments in a way that minimizes

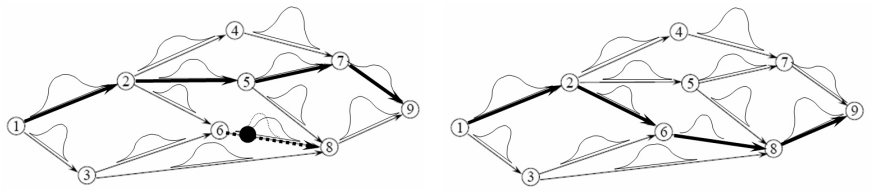
$$\min_{\pi} \mathbb{E} F^{\pi}(c, W) \tag{13.14}$$

where $F^{\pi}(c, W) = V^N$, the final time-N estimate of the length of the shortest path. Equation (13.14) reflects the key difference between learning on a graph and ranking and selection: we measure individual edges ("alternatives") just as before, but now we are no longer interested in finding the single alternative with the best value. We seek to measure alternatives that help us improve our solution to the shortest-path problem.

To see how this can be done, let us examine the impact of a single observation on our solution, that is, the difference between V^n and V^{n+1} brought about by measuring (i, j). For a fixed edge (i, j) define two paths as follows. The path p_{ij}^n is defined to be the shortest path *containing* (i, j), based on the time-n beliefs, whereas $\bar{p}_{ij}^n$ is the shortest path *not containing* (i, j), again according to the time-n beliefs. Both paths are easy to find. For p_{ij}^n, we can run Bellman-Ford to find the shortest path from s to i, and again for the shortest path from i to t. We then use (i, j) to connect these two paths. To find $\bar{p}_{ij}^n$, we merely run Bellman-Ford once on a modified graph with the edge (i, j) removed. Figure 13.6 gives a visual illustration.

Keep in mind that these two paths also depend on our beliefs at time n. When we run Bellman-Ford to find p_{ij}^n and $\bar{p}_{ij}^n$, we use θ^n for the edge costs. The algorithm will also provide us with V_{ij}^n and $\bar{V}_{ij}^n$, the estimated lengths of both paths. A length of a path is simply the sum of the estimates θ_{ij}^n on every edge (i, j) in the path.

The key insight of this section is the following. If we measure (i, j) at time-n, the path p^{n+1} that will be optimal under the time-$(n + 1)$ beliefs – that is, the path that we



(a) (b)

Figure 13.5 The effect of an observation of a single edge (dotted line) on our beliefs about the shortest path (solid line).

will get by running Bellman-Ford with the costs c^{n+1} at time $n+1$ – will necessarily be either p_{ij}^n or $\bar{p}_{ij}^n$. To understand this result, it helps to think in the following way. Suppose that (i,j) is already part of the current optimal path p^n. Then, if we measure it, our beliefs about it will either improve (in which case p^n will become even better than before) or get worse to the point where it is no longer advisable to route a path through (i,j). In the latter case, the best choice among the remaining paths is $\bar{p}_{ij}^n$. Similarly, if (i,j) is not part of p^n, our beliefs about it will either get worse (in which case we will keep the current optimal solution), or get better to the point where we will choose to route our path through (i,j). In the latter case, p_{ij}^n will be the best such route.

This argument allows us to calculate

$$\nu_{ij}^{KG,n} = \mathbb{E}\left[V^n - V^{n+1} \mid S^n, (i^n, j^n) = (i,j)\right], \tag{13.15}$$

the expected improvement contributed to our estimated length of the shortest path by a single observation of (i,j). We reprise the knowledge gradient notation of Chapter 4 because $\nu_{ij}^{KG,n}$ is the exact analog of (4.3), the marginal value of a single observation. We merely replace $\max_{x'} \theta_{x'}^{n+1} - \max_{x'} \theta_{x'}^n$, the improvement in the estimated value of the best alternative, with $V^n - V^{n+1}$, the improvement in the estimated value of our current implementation decision, which is the solution to a shortest-path problem. Our experimental decision can then be written as

$$(i^n, j^n) = \arg\max_{(i,j)} \nu_{ij}^{KG,n},$$

the edge with the highest value of information.

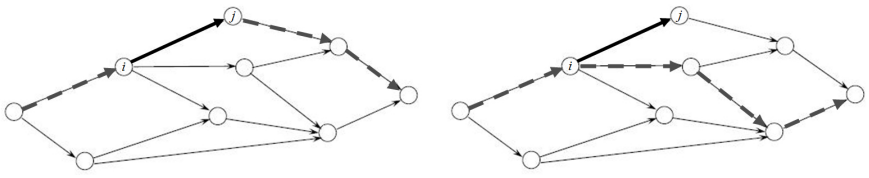
In fact, (13.15) has a closed-form solution

$$\nu_{ij}^{KG,n} = \tilde{\sigma}_{ij}^n f\left(-\frac{\left|V_{ij}^n - \bar{V}_{ij}^n\right|}{\tilde{\sigma}_{ij}^n}\right), \tag{13.16}$$

which is remarkably similar to (4.13), the KG formula for ranking and selection. As before, the quantity

$$\tilde{\sigma}_{ij}^n = \sqrt{\frac{1}{\beta_{ij}^n} - \frac{1}{\beta_{ij}^n + \beta_{ij}^W}}$$

represents the reduction in our uncertainty about (i,j) brought about by a single observation of this edge. The function $f(z) = z\Phi(z) + \phi(z)$ is exactly the same as



(a) (b)

Figure 13.6 Illustration of (a) the best path containing (i,j) and (b) the best path not containing (i,j).

before. The main difference is in the computation of the normalized influence. Where (4.11) uses the quantity $|\theta_x^n - \max_{x' \neq x} \theta_{x'}^n|$, (13.16) uses $\left|V_{ij}^n - \bar{V}_{ij}^n\right|$. This can be thought of as a generalization of (4.11). In ranking and selection, the implementation decision was the same as the experimental decision. We measured individual alternatives in order to find a good alternative, and the influence was determined by the distance between the alternative we chose and the best of the others. In this setting, the influence depends on the distance between the best implementation decision containing the chosen edge (alternative), and the best implementation decision not containing that edge.

13.3 ALTERNATIVE EDGE SELECTION POLICIES

The value of the KG concept, aside from giving us an algorithm for making decisions, is that it allows us to think about new problems. It is not immediately clear how to carry over ideas like Gittins indices, interval estimation, or UCB methods to a problem where the experimental decision is distinct from the implementation decision. On the other hand, the concept of the marginal value of information can be easily extended to such problems. In (13.15), we defined KG as the expected improvement made in our estimate of the value of the optimal implementation decision. We will see in Section 13.4 that this same idea can be used together with more general optimization models.

Still, there are always alternatives. For example, it is possible to shoehorn the graph problem into the framework of ranking and selection, using the concepts we developed in Chapter **??** for subset selection. A shortest-path problem can be viewed as a type of subset selection, because a path is just a subset of the edge set $\mathcal{E}$. We can convert the problem of learning on a graph into a ranking and selection problem where each alternative is a path. The only issue is that a graph is typically described by a set of nodes and edges. The set of all possible paths is typically very large and difficult to enumerate. Fortunately, we can use Monte Carlo simulation to create a small choice set of paths, as in Section 9.2.4.

Given the current belief state (θ^n, β^n), fix an integer K. For every $(i,j) \in \mathcal{E}$, generate samples $\bar{c}_{ij}^n(\omega_k)$, $k = 1, ..., K$ from the prior distribution $\mathcal{N}\left(\theta_{ij}^n, \beta_{ij}^n\right)$. Now, for every $k = 1, ..., K$, run the Bellman-Ford algorithm using the sampled costs $\bar{c}^n(\omega_k)$. The resulting path $\bar{p}_k^n$ is the optimal solution for the kth sample.

Our beliefs about the edges now induce a set of beliefs $\left(\theta^{paths,n}, \Sigma^{paths,n}\right)$ on the K paths we generated, according to the equations

$$\begin{aligned}
\theta_k^{paths,n} &= \sum_{(i,j) \in \bar{p}_k^n} \theta_{ij}^n, \qquad k = 1, ..., K, \\
\Sigma_{k,l}^{paths,n} &= \sum_{(i,j) \in \bar{p}_k^n \cap \bar{p}_l^n} \frac{1}{\beta_{ij}^n}, \qquad k, k' = 1, ..., K.
\end{aligned}$$

We can now apply any standard ranking and selection algorithm (including KG with correlated beliefs!) to this prior. The result will be some path $\bar{p}^*$. We can then use a simple heuristic to pick a single edge to measure from this path. One sensible choice

is

$$(i^n, j^n) = \arg \min_{(i,j)\in\bar{p}^*} \beta_{ij}^n,$$

the edge with the lowest precision on the path of interest. This approach allows us to avoid having to re-derive a KG formula every time we work on a new problem class. Instead, we can simply use the old framework of ranking and selection a little more flexibly.

13.4 LEARNING COSTS FOR LINEAR PROGRAMS*

We now extend the learning problem described in Section 13.2 to the broader setting of a general LP. This section is intended for readers with an interest in more advanced topics. Some familiarity with LP geometry and duality theory will be very helpful for understanding this material; Chvátal (1983) or Vanderbei (2008) are useful LP references.

We return to the optimization problem

$$V(c) = \max_x \; c^T x \tag{13.17}$$

subject to:

$$\begin{aligned} Ax &= b, \\ x &\geq 0, \end{aligned}$$

with which we started this chapter. Recall that, by the strong duality property of linear programming,

$$V(c) = \min_{y,s} \; b^T y \tag{13.18}$$

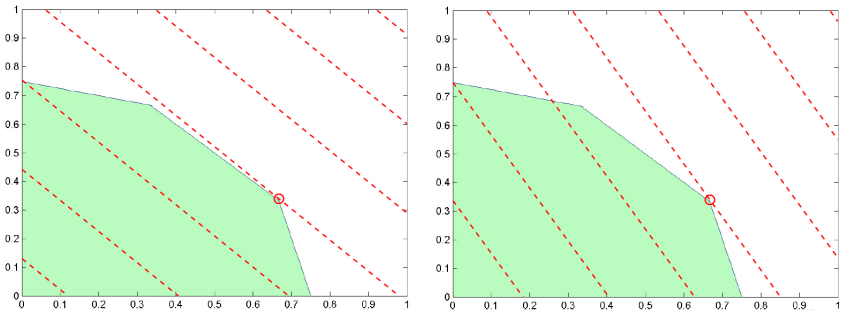
subject to

$$\begin{aligned} A^T y - s &= c, \\ s &\geq 0. \end{aligned}$$

That is, the dual LP has the same optimal value. We can let $x(c)$, $y(c)$ and $s(c)$ represent the optimal primal and dual solutions, that is, the choices of x, y, s in (13.17) and (13.18) that achieve the value $V(c)$.

Now, we assume that c is unknown (the other parameters A and b will still be known), and begin with a multivariate Gaussian prior $c \sim \mathcal{N}\left(\theta^0, \Sigma^0\right)$. As in Section 13.2, we have a budget of N experiments of individual objective coefficients which we can use to improve our solution. Measuring the jth coefficient will give us an observation from the distribution $\mathcal{N}(c_j, \lambda_j)$ where λ_j denotes a known variance. Measuring j^n at time n will cause our beliefs to change according to the equations

$$\begin{aligned} \theta^{n+1} &= \theta^n + \frac{W_{j^n}^{n+1} - \theta_{j^n}^n}{\lambda_{j^n} + \Sigma_{j^n j^n}^n} \Sigma^n e_{j^n}, \\ \Sigma^{n+1} &= \Sigma^n - \frac{\Sigma^n e_{j^n} e_{j^n}^T \Sigma^n}{\lambda_{j^n} + \Sigma_{j^n j^n}^n}, \end{aligned}$$



(a) (b)

Figure 13.7 The optimal solution $V\left(\theta^n + \tilde{\sigma}\left(j\right)z\right)$ remains unchanged over a range of values of z.

exactly as in (2.18) and (2.19). We can see that the graph problem in Section 13.2 is just a special case of this more general problem. The graph structure can be encoded in the matrix A, and we no longer require independent beliefs. Our belief state $S^n = (\theta^n, \Sigma^n)$ consists of the parameters of the multivariate Gaussian distribution.

At time n, our beliefs about the optimal value of the LP are given by $V\left(\theta^n\right)$. We simply solve the deterministic LP (using the simplex method or any standard software package such as Excel) using the vector θ^n for the objective coefficients. Our objective is to choose a learning policy π to maximize

$$\max_{\pi} \mathbb{E} F^{\pi}\left(c, W\right)$$

with $F^{\pi}\left(c, W\right) = V\left(c^N\right)$. The KG factor in this problem can be defined as

$$\nu_j^{KG,n} = \mathbb{E}\left[V\left(\theta^{n+1}\right) - V\left(\theta^n\right) \mid S^n, j^n = j\right]. \tag{13.19}$$

Next, recall from (4.22) that the conditional distribution of θ^{n+1}, given that we measure j at time n, can be written as

$$\theta^{n+1} = \theta^n + \tilde{\sigma}\left(j\right) Z^{n+1},$$

where $Z^{n+1} \sim \mathcal{N}\left(0,1\right)$ and $\tilde{\sigma}\left(j\right) = \frac{\Sigma^n e_j}{\sqrt{\lambda_j + \Sigma^n_{jj}}}$ is a vector of variance reductions. Thus, the challenge of computing the expectation in (13.19) reduces to solving

$$\mathbb{E}\left[V\left(\theta^{n+1}\right) \mid S^n, j^n = j\right] = \mathbb{E} V\left(\theta^n + \tilde{\sigma}\left(j\right) Z^{n+1}\right), \tag{13.20}$$

which is the expected value of a linear program with a stochastic objective function. In general, this problem is computationally intractable. However, in this specific case, the randomness in the objective function depends on a single scalar random variable. The expression $V\left(\theta^n + \tilde{\sigma}\left(j\right) Z^{n+1}\right)$ can be interpreted as the optimal value of an LP whose objective function θ^n is perturbed by a single stochastic parameter Z^{n+1}. If Z^{n+1} were to be replaced by a fixed quantity z, this would be a standard problem in LP sensitivity analysis. In our case, we have to analyze the sensitivity of the LP over the entire distribution of Z^{n+1}.

Let us recall two facts from linear programming. First, the optimal solution always occurs at a corner point (or "extreme point") of the feasible region. Figure 13.7 shows an example in two dimensions where the shaded region represents the set of feasible solutions x. For some fixed value z, the dashed lines in Figure 13.7(a) represent the level curves of the objective function, or points where $\left(\theta^n + \tilde{\sigma}\left(j\right)z\right)^T x = a$ for different values of a. The value of a that makes the level curve tangent to the feasible region is precisely $V\left(\theta^n + \tilde{\sigma}\left(j\right)z\right)$, and the corner point where the level curve touches the feasible region is the optimal solution $x\left(c^n + \tilde{\sigma}\left(j\right)z\right)$.

The second fact is that, for small enough changes in the sensitivity parameter z, the optimal solution $x\left(c^n + \tilde{\sigma}\left(j\right)z\right)$ does not change. In Figure 13.7(b), we change z slightly, rotating all of the level curves. However, the optimal level curve is tangent to the feasible region at the same point as before. The range of values of z for which the optimal solution does not change is sometimes called the *invariant support set*. We can rewrite $\mathbb{E}V\left(\theta^n + \tilde{\sigma}\left(j\right)Z^{n+1}\right)$ as a sum of integrals over the different possible invariant support sets. Suppose that $z_1, ..., z_I$ are a finite set of points, arranged in increasing order, such that $x\left(\theta^n + \tilde{\sigma}\left(j\right)z\right)$ is constant for all $z_i < z < z_{i+1}$. Then, for every i, there is a single corner point x_i satisfying $x_i = x\left(\theta^n + \tilde{\sigma}\left(j\right)Z^{n+1}\right)$ for $z_i < Z^{n+1} < z_{i+1}$. Consequently, taking the expectation over the distribution of Z^{n+1}, we obtain

$$\begin{aligned}\mathbb{E}V\left(\theta^n + \tilde{\sigma}\left(j\right)Z^{n+1}\right) &= \sum_i \int_{z_i}^{z_{i+1}} \left(\theta^n + \tilde{\sigma}\left(j\right)z\right)^T x_i \phi\left(z\right) dz \\ &= \sum_i a_i\left(\Phi\left(z_{i+1}\right) - \Phi\left(z_i\right)\right) + b_i\left(\phi\left(z_i\right) - \phi\left(z_{i+1}\right)\right),\end{aligned}$$

where $a_i = \left(\theta^n\right)^T x_i$ and $b_i = \tilde{\sigma}\left(j\right)^T x_i$. It turns out that this expression is equivalent to

$$\mathbb{E}V\left(\theta^n + \tilde{\sigma}\left(j\right)Z^{n+1}\right) = \left(\max_i a_i\right) + \sum_i \left(b_{i+1} - b_i\right) f\left(-\left|z_i\right|\right),$$

where f is once again the familiar function $f\left(z\right) = z\Phi\left(z\right) + \phi\left(z\right)$. We now make the observation that

$$\max_i a_i = \max_i \left(\theta^n\right)^T x_i = V\left(\theta^n\right),$$

because $\max_i a_i$ is the largest time-n objective value at all of the extreme points that could ever be optimal for any perturbation z, including $z = 0$. Therefore, $x\left(\theta^n\right)$ is one of the points x_i and its value is precisely $V\left(\theta^n\right)$. Therefore,

$$\mathbb{E}V\left(\theta^n + \tilde{\sigma}\left(j\right)Z^{n+1}\right) = V\left(\theta^n\right) + \sum_i \left(b_{i+1} - b_i\right) f\left(-\left|z_i\right|\right). \tag{13.21}$$

Combining (13.19), (13.20) and (13.21) gives us

$$\nu_j^{KG,n} = \sum_i \left(b_{i+1} - b_i\right) f\left(-\left|z_i\right|\right), \tag{13.22}$$

which looks identical to the computation of KG in ranking and selection with correlated beliefs (Section 4.5), with the important difference that the breakpoints z_i now represent the values of Z^{n+1} that change our implementation decision, which in this case is a corner point of the LP.

In the remainder of this section, we explain how the breakpoints z_i and the corresponding corner points x_i can be computed. After this is done, computing the KG factor is a simple matter of applying (13.22). We can find the breakpoints by starting with $x(\theta^n)$, which is the optimal solution when $z = 0$, finding the breakpoints that are nearest to $z = 0$, and expanding outward until all the breakpoints have been found.

Our first order of business is to determine whether $z = 0$ is itself a breakpoint. We can do this by calculating the smallest and largest values of z for which $x(\theta^n)$ is an optimal solution of the perturbed LP. These quantities, which we denote as z_- and z_+, are the optimal values of the LPs

$$z_- = \min_{y,s,z} \quad z \tag{13.23}$$

subject to

$$\begin{aligned} A^T y - s - z\tilde{\sigma}(j) &= \theta^n, \\ x(\theta^n)^T s &= 0, \\ s &\geq 0, \end{aligned}$$

and

$$z_+ = \max_{y,s,z} \quad z \tag{13.24}$$

subject to

$$\begin{aligned} A^T y - s - z\tilde{\sigma}(j) &= \theta^n, \\ x(\theta^n)^T s &= 0, \\ s &\geq 0. \end{aligned}$$

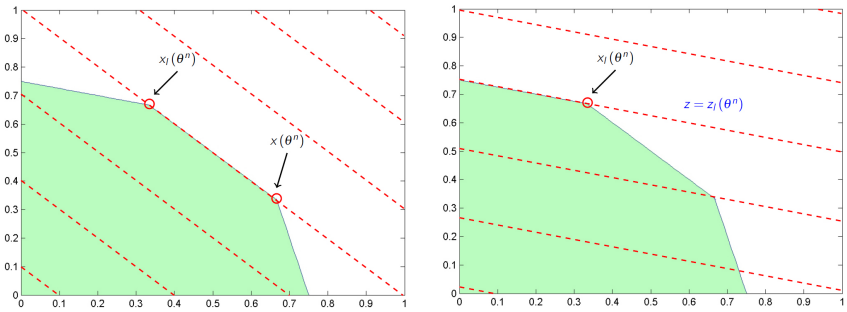
The feasible region, which is the same in both (13.23) and (13.24), merely ensures that $x(\theta^n)$ continues to be an optimal solution of the perturbed LP. Recall that there are three conditions for x to be an optimal solution to the primal LP (13.17) and y, s to be an optimal solution to the dual LP (13.18). First, x has to be primal-feasible. Second, y and s have to be dual-feasible. Third, we must have $x^T s = 0$, known as the complementary slackness property. In our case, $x(\theta^n)$ is automatically feasible for the perturbed LP, because it is optimal for the unperturbed LP

$$V(\theta^n) = \max_x \quad (\theta^n)^T x \tag{13.25}$$

subject to

$$\begin{aligned} Ax &= b, \\ x &\geq 0, \end{aligned}$$

which has the same feasible region as the perturbed LP. The constraints $A^T y - s - z\tilde{\sigma}(j) = \theta^n$ and $x(\theta^n)^T s = 0$ in (13.23) and (13.24) simply ensure that we are able to find a solution to the dual of the perturbed LP that maintains complementary slackness with $x(\theta^n)$. We are looking for the smallest and largest values of the perturbation z for which this is still possible. We now analyze four possible outcomes of this procedure.



(a) (b)

Figure 13.8 When $z_- = 0$, the optimal level curve is tangent to a face of the feasible region. The endpoints of this face are $x(\theta^n)$ and $x_l(\theta^n)$.

Case 1: $\boldsymbol{z_- < 0 < z_+}$ In this case, $z = 0$ is not a breakpoint. This is the case illustrated in Figure 13.7(a). The level curve of θ^n is tangent to a single corner point. However, both z_- and z_+ are breakpoints.

Case 2: $\boldsymbol{z_- = 0, z_+ > 0}$ When zero is a breakpoint, the optimal level curve is tangent to a face of the feasible region, rather than to a single corner point, and the solution $x(\theta^n)$ is located at one corner of the face. Figure 13.8(a) provides a visual illustration. If we apply a positive perturbation $0 < z < z_+$, the level curves will rotate clockwise, but $x(\theta^n)$ will still be the optimal solution. However, any negative perturbation will rotate the level curves counter-clockwise and change the optimal solution.

To find the next breakpoint $z_l(\theta^n) < 0$, we first need to locate the other corner $x_l(\theta^n)$ of the face. This solution will still be optimal for the unperturbed LP, but it will also be optimal for the perturbed LP with $z = z_l(\theta^n)$.

We can do this by solving two LPs. First, $x_l(\theta^n)$ is the optimal solution to the problem

$$V_l(\theta^n) = \min_x \tilde{\sigma}(j)^T x \tag{13.26}$$

subject to

$$\begin{aligned} Ax &= b, \\ s(\theta^n)^T x &= 0, \\ x &\geq 0. \end{aligned}$$

Any feasible solution to this problem continues to be optimal for the unperturbed LP in (13.25). In Figure 13.8(a), such a solution would be located somewhere on the face of the feasible region. Of all such points, the x with the lowest $\tilde{\sigma}(j)^T x$ corresponds to the opposite corner of the face. The optimal value of the LP

$$z_l(\theta^n) = \min_{y,s,z} z \tag{13.27}$$

subject to

$$\begin{aligned} A^T y - s - z\tilde{\sigma}(j) &= \theta^n, \\ x_l(\theta^n)^T s &= 0, \\ s &\geq 0, \end{aligned}$$

corresponds to the next breakpoint, the most negative perturbation for which the perturbed LP still has $x_l(\theta^n)$ as an optimal solution. The relationship between $x(\theta^n)$, $x_l(\theta^n)$, and $z_l(\theta^n)$ can be seen in Figure 13.8(b).

Case 3: $z_- < 0, z_+ = 0$ Zero is still a breakpoint, but $x(\theta^n)$ is now at the opposite corner of the face. We can now rotate the level curves counter-clockwise without changing the optimum, but not clockwise. The next corner point $x_u(\theta^n)$ is the optimal solution to the problem

$$V_u(\theta^n) = \max_x \quad \tilde{\sigma}(j)^T x \tag{13.28}$$

subject to

$$\begin{aligned} Ax &= b, \\ s(\theta^n)^T x &= 0, \\ x &\geq 0. \end{aligned}$$

The next breakpoint $z_u(\theta^n) > 0$ is the optimal value of the LP

$$z_u(\theta^n) = \quad \max_{y,s,z} \quad z$$

subject to

$$\begin{aligned} A^T y - s - z\tilde{\sigma}(j) &= \theta^n, \\ x_u(\theta^n)^T s &= 0, \\ s &\geq 0. \end{aligned}$$

Case 4: $z_- = z_+ = 0$ Zero is a breakpoint, so the optimal level curve for the vector θ^n is still tangent to a face of the feasible region. However, $x(\theta^n)$ is no longer a corner point. Rather, it is somewhere in the middle of the face. We can sometimes discover optimal solutions like this if we use an interior-point algorithm to solve (13.25) instead of the simplex method. Fortunately, this case is easy to deal with: we simply solve (13.26)-(13.29) all together to find both corner points of the face at the same time.

Putting it all together We can repeat this analysis to find sequences of positive and negative breakpoints. For example, if $z_- = 0$ and $z_+ = 0$, as in Case 2, we can find $z_l(\theta^n)$ and then repeat the analysis of Case 2 again, this time with $\theta^n + \tilde{\sigma}(j) z_l(\theta^n)$ as the "unperturbed" objective function, in place of θ^n. This will give us the next negative breakpoint after $z_l(\theta^n)$, and so on, until (13.27) becomes unbounded, which means that the same solution will continue to be optimal regardless of how much we perturb the objective function. The procedure to find a vector $\bar{z}$

of breakpoints and a vector $\bar{x}$ of corresponding corner points can be summarized as follows.

1) Solve (13.25) to obtain $x\left(\theta^n\right)$, $y\left(\theta^n\right)$ and $s\left(\theta^n\right)$.

2) Solve (13.23) and (13.24) to obtain z_-, z_+.

   2a) If $z_- < 0 < z_+$, let $\bar{z} = (z_-, z_+)$ and $\bar{x} = \left(x\left(\theta^n\right)\right)$.

   2b) Otherwise, let $\bar{z} = (0)$ and let $\bar{x}$ be an empty vector.

3) While $\min \bar{z} > -\infty$, do the following:

   3a) Let $\theta' = \theta^n + \tilde{\sigma}(j) \min \bar{z}$.

   3b) Find $x_l\left(\theta'\right)$ and $z_l\left(\theta'\right)$ using (13.26) and (13.27).

   3c) Update $\bar{z} \leftarrow \left(\min \bar{z} + z_l\left(\theta'\right), \bar{z}\right)$ and $\bar{x} \leftarrow \left(x_l\left(\theta'\right), \bar{x}\right)$.

4) While $\max \bar{z} < \infty$, do the following:

   4a) Let $\theta' = \theta^n + \tilde{\sigma}(j) \max \bar{z}$.

   4b) Find $x_u\left(\theta'\right)$ and $z_u\left(\theta'\right)$ using (13.28) and (13.29).

   4c) Update $\bar{z} \leftarrow \left(\bar{z}, \max \bar{z} + z_u\left(\theta'\right)\right)$ and $\bar{x} \leftarrow \left(\bar{x}, x_u\left(\theta'\right)\right)$.

13.5 BIBLIOGRAPHIC NOTES

Section 13.1 - For other important mathematical programming applications (with a focus on the network and LP models considered in this chapter), see e.g. Vanderbei (2008).

Sections 13.2 - 13.3 - These sections are based on Ryzhov & Powell (2011*b*). Prior to this paper, several authors in the computer science community addressed the "bandit shortest path" problem, which would be the online version of the shortest path problem that we consider. For example, Abernethy et al. (2008) describes an algorithm that requires enumerating all the paths, ignoring the structure that arises from overlapping paths. Takimoto & Warmuth (2003) describes the use of a hedging algorithm for the online shortest path problems. Stochastic shortest path problems have been studied even earlier (Kulkarni 1986, Snyder & Steele 1995, Peer & Sharma 2007), but usually with the assumption of a known distribution for the rewards.

Section 13.4 - This material is due to Ryzhov & Powell (2011*a*). The procedure for finding breakpoints comes from Hadigheh & Terlaky (2006).