CHAPTER 2

Notes on Optimization

Rajeev Kohli
Columbia University

Though this be madness, yet there is method in it.
Hamlet II, ii — William Shakespeare

Objective and Scope

Mathematical Programming concerns optimizing a function of many variables, often subject to a set of constraints. It includes linear, nonlinear, integer and stochastic programming. This introductory chapter covers the basics of linear programming and integer programming. It discusses branch-and-bound; dynamic programming; greedy heuristics; worst-case analysis of algorithms; and computational complexity. The material is presented in an informal manner, with an emphasis on insight and intuition. No attempt is made to give an overview of the wide ranging problems and issues in mathematical programming. For example, there is no discussion of convexity and the theory of polyhedra, or of control theory. Applications to two-person, zero sum games, and to several resource allocation problems in marketing are described.

Linear Programming

The following is an example of a linear-programming (LP) problem:

$$\begin{aligned} \text{Maximize} \quad & z = v_1x_1 + \cdots + v_nx_n \\ \text{subject to} \quad & w_1x_1 + \cdots + w_nx_n \leq c \\ & x_i \geq 0, i = 1, \ldots, n. \end{aligned}$$

The first row is the objective function. The next two rows are the constraints. The word "linear" refers to the types of terms in the objective function and the constraints: all of them have the form $a_1x_1 + \cdots + a_nx_n$. The word "programming" refers to "planning." The variables of the problem are called "decision" variables. The above linear programming problem is called the "continuous" knapsack problem (in contrast with the integer knapsack problem, and the 0-1 knapsack problem, which we will consider later). It describes each of the following problems:

1. A consumer with budget c evaluates n products for purchase. The marginal utility for alternative i is a constant value v_i. Item i costs w_i. How should the consumer allocate a budget c to maximize his or her total utility?

2. An expenditure of $\$w_i$ on promotion i increases sales by $\$v_i$. How should a retailer allocate an advertising budget c to maximize the incremental sales?
3. An average item in product category i occupies shelf space w_i and gives an average return of $\$v_i$. How should a retailer allocate a shelf scape c to maximize the total return? across product categories.
4. Security i is priced at $\$w_i$ and has an expected return of $\$v_i$. How should an investor allocate $\$c$ across n securities so as to maximize his or her expected return?

Let $d_i = v_i/w_i$, for all $i = 1, \ldots, n$. We call d_i the density of item i, because it is the amount by which the objective function value increases if we put in one more unit of item i. Without loss of generality, let $d_1 = \max\{d_1, \ldots, d_n\}$; item $i = 1$ has the largest density. Then the optimal solution to the above problem has value $z = cd_1$; and $x_1 = c/w_1, x_2 = \cdots = x_{n+1} = 0$, is an optimal solution. To see why, suppose item $i > 1$ entered the optimal solution. Then we could replace item i by item 1, possibly increasing, but never decreasing, the value of the objective function. Conclusion: if the decision variables are continuous, it is optimal to completely fill the knapsack with the densest item.

As in the above example, all linear programs contain only continuous decision variables. Optimization problems with discrete variables are called discrete-optimization problems, or integer-programming problems, even if the constraints and objective function are linear functions of the variables. There are two good thing about linear programs. First, LP problems are computationally easy, in a sense we will make precise in a later chapter. Second, there are standard methods for solving LP problems. If we can represent a problem as a linear program, we can solve it using a “canned” procedure and be sure that we will obtain a solution within a “reasonable” amount of time, provided a finite solution exists. This characterization of LP problems is an exceptional property. Discrete optimization problems are not so privileged. We often have to design unique algorithms for each such problem. In most cases, the problems are much harder than LP, and we need to use approximate solution procedures. Put another way, we can think our job as almost done if we can formulate a problem as a linear program; and we are only just starting if we can formulate your problem as a discrete optimization problem. For example, the above linear program is very easy to solve if the decision variables are continuous. But it is much harder to solve if we add the restriction that x_i variables are integers, $0, 1, \ldots$. The latter problem is called the integer knapsack problem. It coincides with the problem of filling a knapsack of capacity c with integer units of items, where each unit of item i has size w_i and value v_i, $i = 1, \ldots, n$. If we solve the problem as a linear program, we can get non-integer solutions, which we cannot “round off’ to obtain an optimal solution. We have to device other methods to solve the problem. We will consider discrete optimization problems later in the book. In this chapter, we restrict ourselves to LP problems.

Feasibility and Boundedness. A linear program does not always have a finite solution. It is also possible that a linear program has no feasible solution at all. The first happens when the optimal solution increases (or decreases) without limit. For example, if we maximize $x_1 - x_2$ with the constraints $x_1 \geq 0, x_2 \geq 0$, then there is no finite value of x_1 that maximizes the objective function. Another example:

$$\max z = 7x_1 + 12x_2$$

$$\begin{aligned}\text{Subject to } \quad -2x_1 + 3x_2 &\leq 30\\ -4x_1 + 3x_2 &\leq 12\\ x_1 - 2x_2 &\leq 16\\ x_1 \geq 0, x_2 &\geq 0\end{aligned}$$

There are no feasible solutions when there are conflicting constraints; e.g.,

$$x_1 + x_2 > 5, x_1 > 6, x_1 \geq 0, x_2 \geq 0;$$

or

$$2x_1 + 3x_2 \geq 30,\ x_1 + 2x_2 \leq 16,\ x_1 \geq 0, x_2 \geq 4.$$

It is easy to see that there is no feasible solution here; in large problems with hundreds of variables and constraints, visual inspection can be difficult. Finding out if an LP model has a feasible solution is essentially as hard as actually finding the optimal LP solution.

Solution Procedures. One good thing about linear programs is that there are well known methods for solving them, regardless of how many variables and constraints there might be in the problem — assuming that a solution exists and is bounded, by which we mean that it does not go to plus or minus infinity. Linear program is called an "easy" problem because there are solution procedures that are efficient in the sense that as the number of problem parameters — the number of variables and constraints — increase, the time to solve the problem increases at a "reasonable" rate, by which we mean a rate that is a polynomial function of the number parameters. We will discuss what we mean by efficient in a separate chapter. For the present, we note that we are dealing with an easy problem if we can formulate it as a linear program. We can use well-developed methods for solving the problem. These methods are described very well in many books, for example those by Chvatal(1983)[1] and Bertsimas and Tsitsiklis (1997).[2] For this reason, we will focus more on stating some important properties of linear programs and then discuss marketing applications of linear programming.

The feasible set of solutions for a linear program, when it exists, is a polyhedron. It is a space bounded by "straight edges." If we have only two decision variables, then the feasible set of solution is bounded by intersecting straight lines in a two-dimensional plot showing the values of the variables on the two axes. If we have three decision variables, and we plot the values of the variables along three dimensions, then the feasible region, when it exists, is bounded by intersecting planes. Points in the space enclosed by and on the planes are feasible solutions. The mathematical generalization of intersecting lines and planes are called facets; and the points contained within and on such intersecting facets form a convex polyhedral. We can't visualize them but can imagine them and write them in mathematical terms. If you take any two points in a convex polyhedral, you can join them by a line that is entirely in the set. A convex set is not a good place for children playing hide and seek — you can "see" every point from every other point in the set.[3]

An optimal solution to an LP problem is a "corner-point" of the polyhedron that describes the feasible region. That is, it is not a point in the interior of the

[1]Chvatal, Vasek (1983), *Linear Programming,* " New York: W. H. Freeman.

[2]Bertsimas and Tsitsiklis (1997), *Introduction to Linear Optimization,* Belmont, MA.: Athena Scientific.

[3]This colorful description is from Chvatal (1983); see footnote 1.

polyhedral; and it is a point at which the facets intersect. Note that the solution may not be unique; all we can say is that a corner point is an optimal solution, and that points inside the feasible polyhedron are not optimal solutions. But it is possible to have all the points on a facet be an optimal solution; when this occurs, we say that the solution to a linear program is *degenerate.*

One might be tempted to restrict search to these intersecting points when looking for an optimal solution. This is a good idea, but it has the following problem. There can be too many corner points. So we have to use methods that can efficiently search among the corner point solutions. The Simplex method does this by going from one corner point to the next, all the while improving the solution value. Newer methods, called barrier or interior-point methods, visit points within the interior of the feasible region. These methods derive from techniques for nonlinear programming that were developed and popularized in the 1960s by Fiacco and McCormick. Their application to linear programming dates back to Karmarkar's innovative work in 1984. Bertsimas and Tsitsiklis (1997) give a good description of these solution procedures.

LP Dual. An important thing to know about linear programs is that each of them has an associated "dual" problem. It is also a linear program, and it has the same objective function value as the original — so-called "primal" — problem. We defer the discussion of the general case to the next section. Here, we illustrate the key ideas here using the above (continuous knapsack) problem as an example.

Suppose we can have lower and upper bounds — say, $z_{\min}$ and $z_{\max}$ — for the objective function. That is , $z_{\min} \leq z \leq z_{\max}$. The smaller the gap between $z_{\min}$ and $z_{\max}$, the narrower the range within which the objective function lies; and if $z = z_{\min} = z_{\max}$, then we know that the value of z is the value of the optimal solution.

Let us use this approach to find the solution to the continuous knapsack problem. We first construct an upper bound. We multiply both sides of the constraint $w_i x_1 + \dots w_n x_n \leq c$, by some $y > 0$ to obtain

$$y(w_1 x_1 + \cdots + w_n x_n) \leq yc.$$

We want to use the right-hand side, yc, as an upper bound on the value of the objective function. This can be justified only if for all $i = 1, \dots, n$, the coefficient yw_i for variable x_i is as large as the coefficient v_i that appears with x_i in the objective function. That is, we can write

$$v_1 x_1 + \cdots + v_n x_n \leq y(w_1 x_1 + \cdots + w_n x_n) \leq yc,$$

only if

$$w_i y \geq v_i, \quad \text{for all } i = 1, \dots, n.$$

Finding the tightest upper bound thus requires that we minimize cy subject to the constraint $w_i y \geq v_i$:

$$\begin{aligned} \text{Minimize} \quad & cy \\ \text{subject to} \quad & w_i y \geq v_i, \ i = 1, \dots, n \\ & y \geq 0. \end{aligned}$$

This is called the dual problem. We write down its solution immediately: $y = \max_i v_i / w_i = d_1$. Thus, we have the upper bound $z_{\max} = cd_1$. We now obtain a lower bound. If we fill the entire knapsack with c/w_i units of item i, we get a solution value of $cv_i/w_i = d_i c$. As $d_1 = \max_i d_i$, we obtain the lower bound $z_{\min} = cd_1$, when we fill the knapsack with item 1. As $z_{\min} = z_{\max} = cd_1$, we conclude that $z^* = cd_1$ is the optimal solution value and that $x_1 = c/w_1, x_2 = \ldots x_n = 0$, is the optimal solution.

We can also proceed in the reverse direction. That is, suppose have to solve the above minimization problem, which we called the dual problem. If we write the dual for the dual problem, we will get the original, primal problem. To see how, we multiply both sides of the constraint $w_i y \geq v_i$ by $x_i \geq 0, i = 1, \ldots, m$. We then add both sides to get

$$x_1 v_1 + \cdots + v_n x_n \leq (w_1 x_1 + \cdots + w_n x_n) y.$$

We want to use the left-hand side as a lower bound on the value of the objective function. This can be justified if the right-hand side is no smaller than cy; i.e., if

$$w_1 x_1 + \cdots + w_n x_n \leq c.$$

Finding the tightest (highest) lower bound thus requires that we solve the following problem:

$$\begin{aligned} \text{Maximize} \quad & z = v_1 x_1 + \cdots + v_n x_n \\ \text{subject to} \quad & w_1 x_1 + \cdots + w_n x_n \leq c \\ & x_i \geq 0, \; i = 1, \ldots, n. \end{aligned}$$

This is the original linear program.

Shadow prices and complementary slackness. Suppose we change the capacity of the knapsack from c to $c + \epsilon$ units. As we will still fill the knapsack with the densest item, the value of the optimal solution will increase by $\delta = d_1 \times \epsilon$. Put another way, $\delta/\epsilon = d_1$ is the rate of increase in the objective function value if we increase the right hand side of the capacity constraint by an arbitrarily small amount. Recall that in the knapsack problem, the dual variable y has the value

$$y = \max\{d_1, \ldots, d_n\} = d_1.$$

So

$$y = \delta/\epsilon.$$

This is not a coincidence. It always happens that the value of a dual variable is equal to rate at which the objective function changes in response to a marginal change in the right hand size of a (related) constraint. For this reason, the values of a dual variable is called a shadow prices. It is the maximum (marginal) price you would pay to relax a constraint by a small amount. For example, if the right hand side of a constraint gives the maximum available promotion budget, then the value of the associated dual variable tells you the amount by which you can increase the objective function — say, sales or profit — if you increase the budget by a penny.

If the value of the slack variable is zero, then you will pay nothing to relax the constraint. Otherwise you will. Complementary slackness refers to the following property of a linear program. If a constraint has slack is not satisfied as an equality by an optimal solution, then the associated dual variable has a value of zero; otherwise, it has a non-zero value. That is, a dual variable has a value of zero if there is "slackness" in the associated constraint in the sense that it is not satisfied as an equality. It is not zero if the constraint is "tight" — it is satisfied as an equality.

Generalization. We now generalize the above approach for any linear program. Although the solution is no longer as easy to find, we still obtain the condition that the original linear program — called the primal problem — has the same objective function value as the solution to a dual problem, which is obtained by using the constraints to write down an an upper bound for the primal problem. The complementary slackness conditions still hold; and the interpretation of the dual variables as shadow prices is unchanged.

We can write any linear program as the following problem:

$$\begin{aligned} \text{Maximize} \quad & c_1x_1 + \cdots + c_nx_n \\ \text{subject to} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m. \end{aligned}$$

We will call this the standard (or canonical) form of a linear program. We can represent any minimization problem as the above maximization problem by multiplying the objective function by -1. Similarly, we can represent any $\geq$ constraint by an equivalent $\leq$ constraint by multiplying both sides by -1.

We multiply the ith constraint of the above linear program by $y_i \geq 0,\ i = 1, \ldots, m$. We then add the inequalities to obtain

$$y_1(a_{11}x_1 + \cdots + a_{1n}x_n) + \cdots + y_n(a_{m1}x_1 + \cdots + a_{mn}x_n) \leq b_1y_1 + \ldots b_my_m.$$

We re-group the terms on the left hand side, collecting the x_i terms, to write

$$x_1(a_{11}y_1 + \cdots + a_{m1}y_n) + \cdots + x_1(a_{1n}y_1 + \cdots + a_{mn}y_n) \leq b_1y_1 + \ldots b_my_m.$$

We compare the left-hand side above with the objective function and observe that $b_1y_1 + \ldots b_my_m$ is an upper bound on the objective function value if $a_{1j}y_1 + \cdots + a_{mj}y_n \geq c_j$, for all $j = 1, \ldots, n$. To obtain the tightest upper bound, we seek values $y_i \geq 0$ for which $b_1y_1 + \ldots b_my_m$ is as small as possible. That is, we want to solve the following problem:

$$\begin{aligned} \text{Minimize} \quad & b_1y_1 + \ldots b_my_m \\ \text{subject to} \quad & a_{11}y_1 + \cdots + a_{m1}y_n \geq c_1 \\ & \vdots \\ & a_{1n}y_1 + \cdots + a_{mn}y_n \geq c_n \\ & y_i \geq 0,\ i = 1, \ldots, m. \end{aligned}$$

This is the dual problem; it is also a linear program. It is a theorem of linear programming that the objective function values are the same for the dual problem and the original, primal problem. This is called the duality theorem (or the theorem of strong duality). We state it below; its proof is not given here, because it is typically given in the context of the Simplex method, which is one way of solving a linear program.

THE DUALITY THEOREM. If a linear program has an optimal solution, so does its dual, and their optimal solution values are equal.

A consequence of the duality theorem is that a primal problem has an optimal solution if and only if the dual problem has an optimal solution. If the primal is unbounded, the dual must be infeasible; and if the dual is unbounded, the primal must be infeasible. It is also possible that both the primal problem and the dual problem are infeasible. For example, the following problem is infeasible, as is its dual:

$$\begin{aligned} \text{Maximize } \ & 2x_1 - x_2 \\ \text{subject to } \ & x_1 \le 1 + x_2 \\ & x_1 \ge 2 + x_2 \\ & x_1, x_2 \ge 0. \end{aligned}$$

Complementary slackness. The optimal solutions to the primal and dual problems satisfy the following conditions:

$$x_i > 0 \text{ implies } \sum_{j=1}^{n} a_{ij} y_j = c_i, \text{ for all } i = 1, \ldots, m;$$

$$y_j > 0 \text{ implies } \sum_{i=1}^{m} a_{ij} x_i = b_i, \text{ for all } j = 1, \ldots, n.$$

These conditions are necessary and sufficient for optimality.

One common interpretation of the LP problem and its dual is as follows. An LP finds a set of activities x so that the profit Cx from the activities is maximized; the constraint $AX \le B$ ensures that no resource is over-utilized. The dual finds a set of prices Y for each resource so that the total price BY is minimized; the constraint $A^T Y \ge C$ ensures that the price is high enough to cover the cost for each activity using a resource.

Some Marketing Applications of Linear Programming

Two-Person, Zero-Sum Games. Let R (row) and C (column) denote two players (firms) engaged in a "zero-sum" game, in which the amount one player wins is the amount the other player loses. For example, consider two firms fighting for market share. The gain in market share for one firm is the loss in market share for the other firm.

Player R has m possible strategies, denoted $i = 1, \ldots, m$. Player C has n possible strategies, denoted $j = 1, \ldots, n$. We denote by a_{ij} the payoff for player R if he chooses strategy i and C chooses strategy j. A negative value of a_{ij} means that R loses market share, a positive value means that he wins market share. We represent the payoffs (changes in market shares) in a $m \times n$ matrix A with ijth element a_{ij}, $i = 1, \ldots, m$; $j = 1, \ldots, n$. Row i of A corresponds to the ith strategy for player R; and column j corresponds to the jth strategy of player C.

Let $x = (x_1, \ldots, x_m)$, where x_i denote the probability that player R chooses strategy i, $x_1 + \cdots + x_m = 1$. Similarly, let $y = (y_1, \ldots, y_n)$, where y_j denote the probability that player C chooses strategy j, $y_1 + \cdots + y_n = 1$. Then the expected payoff for player R is

$$xAy = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_i y_j.$$

We can reverse the order of summation and re-group the terms as follows:

$$xAy = \sum_{j=1}^{n} y_j \left(\sum_{i=1}^{m} a_{ij} x_i \right) = \sum_{j=1}^{n} y_j t_j,$$

where

$$t_j = \sum_{i=1}^{m} a_{ij} x_i.$$

Let $t = \min_j t_j$. Then

$$xAy = \sum_{j=1}^{n} y_j t_j \geq \sum_{j=1}^{n} y_j t = t \sum_{j=1}^{n} y_j = t.$$

Thus, player R is assured of a minimum payoff

$$\min_y xAy = t.$$

On the other hand, as each "pure strategy" in which C always uses strategy j is a candidate for minimizing xAy, we have

$$\min_y xAy \leq \sum_{i=1}^{m} a_{ij} x_i \text{ for all } j = 1, \ldots, n.$$

We conclude that among the most effective responses y for player C to the mixed strategy x of player R, there is always at least one pure strategy. Thus,

$$\min_y xAy = \min_j \sum_{i=1}^{m} a_{ij} x_i.$$

We can therefore formulate the problem for player R as follows:

$$\begin{aligned} \text{Maximize} \quad & \min_j \sum_{i=1}^{m} a_{ij} x_i \\ \text{subject to} \quad & \sum_{i=1}^{m} x_i = 1 \\ & x_i \geq 0, \text{ for all } i = 1, \ldots, m. \end{aligned}$$

This problem can be represented as the following linear program, denoted P :

$$\begin{aligned} \text{Maximize} \quad & z \\ \text{subject to} \quad & z \leq \sum_{i=1}^{m} a_{ij} x_i, \text{ for all } j = 1, \ldots, n \\ & \sum_{i=1}^{m} x_i = 1 \\ & x_i \geq 0, \text{ for all } i = 1, \ldots, m. \end{aligned}$$

Similarly, if player C chooses strategy j with probability y_j, his expected loss is at most

$$\max_x xAy,$$

where the maximum above is taken over all possible probability vectors x. Following the same line of reasoning above, we can show that

$$\max_x xAy = \max_i \sum_{j=1}^n a_{ij} y_i.$$

Thus, the problem of finding the optimal strategy for player C can be written as

$$\begin{aligned} &\text{Minimize} \quad \max_i \sum_{j=1}^n a_{ij} y_j \\ &\text{subject to} \quad \sum_{j=1}^n y_j = 1 \\ &\qquad\qquad y_j \geq 0, \ \text{for all } j = 1, \dots, n. \end{aligned}$$

This problem can be represented as the following linear program, denoted D :

$$\begin{aligned} &\text{Maximize} \quad w \\ &\text{subject to} \quad w \geq \sum_{j=1}^n a_{ij} y_j, \ \text{for all } i = 1, \dots, m \\ &\qquad\qquad \sum_{j=1}^n y_j = 1 \\ &\qquad\qquad y_j \geq 0, \ \text{for all } j = 1, \dots, n. \end{aligned}$$

We observe that P and D are LP duals; the mixed strategies for the two players of a zero-sum game are obtained by solving a LP problem and its dual. The duality theorem implies the following result.

THE MINIMAX THEOREM. For every $m \times n$ matrix A there is a stochastic row vector x^* and a stochastic column vector y^* such that

$$\min_y x^* A y = \max_x xAy^*$$

with the minimum taken over all stochastic vectors y and the maximum taken over all stochastic vectors x.

Regression Problems. Let

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \epsilon,$$

where y is a dependent variable, $x_1, \dots, x_n$, are independent variables, and ϵ is an error term. Typically, we obtain estimates of the parameters by minimizing the sum of squared errors. Here, we examine a different approach, using linear programming. Suppose we have m observations, $y_1, \dots, y_m$. We write

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in} + \epsilon_i.$$

The quantity

$$\left(\sum_{i=1}^m |e_i|^p \right)^{1/p}, \quad p \geq 1,$$

is called the L_p-*norm* and is denoted by $||e||_p$. The limit when p becomes arbitrarily large is the L_∞-*norm*:

$$\lim_{p\to\infty} ||e||_p = \max |e_i|.$$

The best L_∞-*norm* approximation is one for which the estimates $\beta_i, i = 0, \ldots, n$ minimize the maximum error; the estimates for the best L_1-*norm* approximation minimize the mean absolute error; and the estimates for the best L_2-*norm* approximation minimize the mean squared error. The last is obtained using calculus; it is also the maximum likelihood estimator of the parameters for independent, Gaussian errors. The best L_1-*norm* and L_∞-*norm* approximations can be obtained in the following manner using linear programming.

Consider the linear program

$$\begin{aligned} \text{Minimize}\quad & e_1 + \cdots + e_n \\ \text{subject to}\quad & e_i \geq y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in}),\ i = 1, \ldots, m \\ & e_i \geq -y_i + (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in}),\ i = 1, \ldots, m. \end{aligned}$$

That is, e_i is at least as large as the absolute difference between the value of y_i and the value of $\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in}$. The objective function sets each e_i to its minimum possible value, and thus it minimizes the sum of absolute errors. If we divide the objective function value by m, we obtain the mean absolute error. Thus, we see that we can use linear programming to solve the problem of estimating the parameters of a linear regression, using the objective of minimizing the mean absolute deviation between the actual and predicted values of the dependent variable.

Now consider the objective of minimizing the maximum error. We can formulate this problem as

$$\begin{aligned} \text{Minimize}\quad & e \\ \text{subject to}\quad & e \geq y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in}),\ i = 1, \ldots, m \\ & e \geq -y_i + (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in}),\ i = 1, \ldots, m. \end{aligned}$$

The constraints imply that the lower bound on e is the maximum absolute error across the m observations. The objective function minimizes the value of this maximum error.

Banner Advertising. Many web sites earn revenue by hosting banner advertisements. For example, each time a visitor to a web site clicks on a banner ad, the sponsor of the ad pays the web site a fixed free. Large sites host thousands of banners each day. These sites have to decide which banner ad to place on which web page. Chickering and Heckerman (2003) describe the following method for banner assignment when the objective is to maximize the total number of click throughs over a period of time (an hour, a day, or a week). Microsoft uses this method for placing banner ads on its MSN network.[4]

The MSN web site contains many different web pages. These pages are grouped into segments: *News, Look it Up, Living, Finances, Entertainment, Shopping,* and *Technology.* The segments are relatively stable, although their content changes

[4]Chickering, M. and D. Heckerman (2003), "Targeted Advertising on the Web with Inventory Management," *Interfaces,* 33 (5), 71–77.

frrequently, sometimes more than once a day. Let n denote the number of segments. We will assume that these are fixed for the period of time T (e.g., a day, week, or month) over which the advertising is to be scheduled. Let m denote the number of distinct banner ads that are to be scheduled over time T. Over a test period preceding the planning period, MSN places each banner advertisement in each segment of the web site. It tracks visits and banner clicks during the test period, from which it estimates the probability p_{ij} than a visitor to segment j clicks on banner ad i, $i = 1, \dots, m$, $j = 1, \dots, n$. The value of p_{ij} is treated as a known value in the planning of banner-ad placement.

Let n denote the number of segments and let m denote the number of banner ads to be scheduled over a period of time T (e.g., a day, week, or month). Before deciding on the placement of banners, MSN conducts a test, placing each banner advertisement in each segment of the web site. It tracks visits and banner clicks during the test period, from which it estimates the probability p_{ij} than a visitor to segment j clicks on banner advertisement i, for all $i = 1, \dots, m$, and $j = 1, \dots, n$. The value of p_{ij} is treated as a known value in the planning of banner-ad placement.

Let x_{ij} denote the number of impressions of advertisement i to be shown in segment j in time T. MSN's objective is to maximize the total number of banner click throughs, because this determines the revenue they earn from posting banner ads on their web site. There are two main constraints. First, each advertisement i has a quota q_i, which is the minimum number of visitors who are exposed to it over the time period T. Second, each segment j of the web site has a capacity s_j; it is the maximum number of all impressions that can be shown in segment j over the time period T.

We can formulate MSN's problem of maximizing click throughs as the following optimization problem:

$$\begin{aligned} \text{Maximize} \quad & z = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{m} x_{ij} \geq q_i, \text{ for all } i = 1, \dots, m \\ & \sum_{i=1}^{n} x_{ij} \leq s_j \text{ for all } j = 1, \dots, n \\ & x_{ij} \geq 0 \text{ for all } i = 1, \dots, m, \; j = 1, \dots, n. \end{aligned}$$

The objective function maximizes the expected number of click throughs across all segments and banner advertisements. The first constraint requires that the number of impressions must exceed the quota for each advertisement over the time period T. The second constraint requires that the total number of impressions must not exceed the capacity s_j for each segment $j = 1, \dots, m$. The last constraint restricts the number of impressions x_{ij} for any advertisement i and segment j to non-negative values. In fact, x_{ij} must be an integer, $0, 1, 2, \dots$. We will assume it is a continuous variable because typically each x_{ij} has a large value and we can then round fractional solutions for x_{ij} to the nearest integer.

The Dual Problem. We relax the constraints to formulate the problem:

$$\min_{y_i,t_j} \quad \max_{x_{ij}} \quad z = \sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}x_{ij} \quad + \sum_{i=1}^{m} y_i(\sum_{j=1}^{m}(x_{ij} - q_i) + \sum_{j=1}^{n} t_j(\sum_{i=1}^{n}(-x_{ij} + s_j)$$

$$\text{subject to}: \quad y_i, t_j \geq 0.$$

This is the same as

$$\min_{y_i,t_j} \quad \max_{x_{ij}} \quad z = \sum_{i=1}^{n}\sum_{j=1}^{m}(p_{ij}x_{ij} + y_i(x_{ij} - q_i) + t_j(-x_{ij} + s_j)$$

$$\text{subject to}: \quad y_i, t_j \geq 0.$$

We can change the order of minimization and maximization to get

$$\max_{x_{ij}} \min_{y_i,t_j} \sum_{i=1}^{n}\sum_{j=1}^{m} -y_i q_i + t_j s_j + x_{ij}(p_{ij} + y_i - t_j)$$

$$\text{subject to}: \quad y_i, t_j \geq 0.$$

Observe that this is the relaxation of the linear program

$$\min_{y_i,t_j} \sum_{i=1}^{n}\sum_{j=1}^{m} -y_i q_i + t_j s_j$$

$$\begin{aligned}\text{subject to}: \quad & p_{ij} + y_i - t_j \geq 0 \\ & y_i, t_j \geq 0.\end{aligned}$$

Customizing E-mail. Some web sites send targeted emails to subscribers. In the simplest case, each email has the same list of n hyperlinks, but the ordering of the hyperlinks varies from one email to another. Let p_{ij} denote the probability that a specific person will click on hyperlink i in position j in an email. Ansari and Mela (2003) describe a method for estimating the values of p_{ij} for each of a list of subscribers.[5]

We can represent the e-customization problem as the following *assignment problem*: given n "jobs" (hyperlinks) and n "workers" (positions in an email), find a least-cost assignment of jobs to workers. Let $c_{ij} = 1 - p_{ij}$ denote the cost of assigning hyperlink i to position j. Then the linear assignment problem is:

$$\begin{aligned}\text{Minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1, \text{ for all } 1 \leq j \leq n \\ & \sum_{j=1}^{n} x_{ij} = 1, \text{ for all } 1 \leq i \leq n \\ & x_{ij} \geq 0, \text{ for all } 1 \leq i \leq n.\end{aligned}$$

[5]Ansari, A. and C.F. Mela (2003), "E-Customization," *Journal of Marketing Research,* 40 (2), 131–145.

The following theorem guarantees that a linear-programming solution to the problem gives an integer solution to the assignment problem. For a proof, we refer the reader to Chvatal (1983); see footnote 1.

Birkhoff-von Neumann theorem. All extreme points of the polyhedron described by the above linear program are 0-1–valued.

The assignment problem section is an example of an integer-programming problem, in which the decision variables — the x_{ij} — can only have integer values. Problems in which the values of the decision variables can be either zero or unity are called $0-1$ integer programs.

Not all integer programs are as easy to solve — it is indeed a fortunate circumstance that the assignment problem can be solved using linear programming. As a class, integer-programming problems are much harder to solve than linear programming, which is considered an easy problem. To better understand why this is so, we discuss the idea of computational complexity in the following section. After that, we consider integer-programming problems, and their marketing applications.

Computational Complexity

The term "computational complexity" is used in two ways. One refers to an algorithm for solving instances of a problem: broadly stated, the computational complexity of an algorithm is a measure of how many steps the algorithm will require in the worst case for an instance or input of a given size. The number of steps is measured as a function of that size. The term's second, more important use is in classifying problems according to their tractability or intractability — whether they are "easy" or "hard" to solve.

To understand what is meant by the complexity of an algorithm, we must define algorithms, problems, and problem instances. We must understand how one measures the size of a problem instance and what constitutes a "step" in an algorithm.

A *problem* is an abstract description coupled with a question; for example, the integer knapsack problem is: "Given a knapsack with capacity c and n items with weights w_i and value $v_i, i = 1, 2, 3, , n$, how many units of item i should be placed in the knapsack to maximize the total value of its contents?"

An *instance* of a problem, on the other hand, includes an exact specification of the data: for example, "The capacity of the knapsack is $c = 12$ and there are five items available with weights and values $\ldots$," and so on.

In mathematical terms, a problem can be thought of as a function p that maps an instance x to an output $p(x)$ (an answer).

An *algorithm* is a set of instructions guaranteed to find the solution for any problem instance in a finite number of steps. In other words, an algorithm is a finite procedure for computing $p(x)$ for any given input x for a problem p. Computer scientists model algorithms by a mathematical construct called a Turing machine. We will consider a slightly different model.

In a simple model of a computing device, a "step" consists of one of the following operations: addition, subtraction, multiplication, finite-precision division, and comparison of two numbers. Thus if an algorithm requires one hundred additions and 220 comparisons for some instance, we say that the algorithm requires 320 steps on that instance. In order to make this number meaningful, we would like to express it as a function of the size of the corresponding instance; but determining the exact function can be impractical. Instead, we formulate a simple function of the input size that is a reasonably tight upper bound on the actual number of steps. Such a function is called the complexity or running time of the algorithm.

Technically, the *size* of a problem instance is the number of bits required to encode it. It is measured in terms of the inherent dimensions of the instance (such as the knapsack capacity and the number of items), plus the number of bits required to encode the numerical information in the instance (such as the item weights). Since numerical data are encoded in binary, an integer C requires about $\log_2 |C|$ bits to encode and so contributes logarithmically to the size of the instance. The running time of the algorithm is then expressed as a function of these parameters, rather than the precise input size. For the integer knapsack problem, an algorithm's running time is expressed as a function of the number of items and the maximum number of bits required to encode the capacity, item values and item sizes.

In analyzing the inherent tractability of a problem, we are interested in how the running time grows with the size of the instance. For two functions $f(t)$ and $g(t)$ of a nonnegative parameter t, we say that $f(t) = O(g(t))$ if there is a constant $c > 0$ such that, for all sufficiently large $t, f(t) \leq cg(t)$. The function $cg(t)$ is thus an asymptotic upper bound on f. For example, $100(t^2 + t) = O(t^2)$, since by taking $c = 101$ the relation follows for $t \geq 100$; however, $0.0001t^3$ is not $O(t^2)$. Notice that it is possible for $f(t) = O(g(t))$ and $g(t) = O(f(t))$ simultaneously.

We say that an algorithm runs in *polynomial time* (is a polynomial-time algorithm) if the running time $f(t) = O(P(t))$, where $P(t)$ is a polynomial function of the input size. Polynomial-time algorithms are considered efficient. Problems for which polynomial time algorithms exist are considered "easy."

The classes P and NP. A decision (or recognition) problem is a problem with a "yes" or "no" answer. For example, the question "is there a budget allocation that guarantees N gross rating points" is a decision problem. A decision problem has an input, in this case the size of the budget and the costs and rating points for each media option. Optimization problems are not recognition problems, but most have recognition counterparts. In general, an optimization problem is not much harder to solve than the associated decision problem. One can usually embed an algorithm for the decision problem in a binary search over the possible objective function values to solve the optimization problem with a polynomial number of calls to the embedded algorithm.

The class P is defined as the set of decision problems for which there exists a polynomial-time algorithm, where " P" stands for "polynomial time;" P comprises those problems that are formally considered "easy." Wood and Dantzig (1949) were the first to solve linear programs, using the simplex method. For the longest time, it was not clear if it was solvable in polynomial time. Khachian answered the question in 1979 by presenting a polynomial-time algorithm for linear programming. A much more efficient polynomial-time algorithm (called an "interior point method") was found by Karmarkar in 1984. So linear programming is "easy," even though to most of us it seems to be the punishment for not having eaten our breakfasts when young.

A larger problem class — NP — contains the class P. The term "NP " stands for "nondeterministic polynomial" and refers to a different, hypothetical model of computation, which can solve the problems in NP in polynomial time. The class NP consists of all recognition problems with the following property: for any "yes" instance of the problem there exists a polynomial-length "certificate" or proof of this fact that can be verified in polynomial time. As an example, consider the recognition version of the integer knapsack problem. Suppose someone claimed that a particular instance has a solution with a value at least 100 and handed us a list showing how many units of each item to put in the knapsack. This list is the certificate: it is polynomial in length, and we can easily verify, in polynomial time, that the solution does in fact produce a solution value no smaller than100.

There is an asymmetry between "yes" and "no" answers — for example, there is no obvious, succinct way for someone to convince us that a particular instance of the integer knapsack problem *not* contain a solution with value at most 100. By reversing the roles played by "yes" and "no" we obtain a problem class known as Co-NP. For every recognition problem in NP there is an associated recognition problem in Co-NP obtained by framing the NP question in the negative; e.g., "do all solutions for a given knapsack problem have a solution no larger than K?" Many recognition problems are believed to lie outside both of the classes NP and Co-NP, because they seem to possess no appropriate "certificate." An example is the question, "Is the number of distinct solutions to the integer knapsack problem with solution value at most K exactly equal to L?"

NP-complete problems. To date, no one has found a polynomial-time algorithm for the integer knapsack problem (the dynamic program we studied appears to be a polynomial time algorithm, but as we will see below, it is not; it pretends to look like one, so we call it a pseudo-polynomial time algorithm). In this sense, it is just like the traveling salesman problem, which I am sure you have heard about. But there is another relation between the two problems. If there is a polynomial-time algorithm for either problem, then there is a polynomial-time algorithm for both. Indeed, there would be a polynomial time algorithm for every problem in NP. In this sense, the integer knapsack problem and the traveling salesman problem are the hardest problems in the class NP. This is a very strong statement: the class NP includes a large number of problems that appear to be extremely difficult to solve. If we were to solve any one of these problems (not in NP but in a subset of NP), we would know that all problems in NP can be solved in polynomial time. This subset of problems in NP contains many other problems besides the integer knapsack problem and the traveling salesman problem. It has a name — it is called NP-complete. People who know a lot more about these things than me think it highly unlikely that a polynomial algorithm will be found for any NP-complete problem, since such an algorithm would actually provide polynomial time algorithms for every problem in NP !

The term *NP-hard* refers to any problem that is at least as hard as any problem in NP. In particular, optimization problems whose recognition versions are NP-complete are NP-hard, since solving the optimization version is at least as hard as solving the recognition version.

The classes NP and NP-complete were introduced by Cook in 1971. In that paper, he demonstrated that a particular recognition problem from logic, SATISFIABILITY, was NP-complete; he showed how every other problem in NP could be encoded as an appropriate special case of SATISFIABILITY. After this, many other problems - traveling salesman, integer and 0-1 knapsack — were shown to be NP-complete. To do so requires simply providing a polynomial transformation from a known NP-complete problem to the candidate problem. Essentially, one needs to show that the known "hard" problem, such as SATISFIABILITY, is a special case of the new problem. Thus, if the new problem has a polynomial-time algorithm, then the known hard problem has one as well.

The *polynomial hierarchy* refers to a vast array of problem classes both within and beyond NP and Co-NP. There are analogous definitions focusing on the space required by an algorithm rather than the time. There are complexity classes for parallel processing, based on allowing a polynomial number of processors. There are classes corresponding to randomized algorithms, those that allow certain decisions in the algorithm to be made based on the outcome of a "coin toss." There are also complexity classes that capture the notions of optimization and approximability.

The most famous open question concerning the polynomial hierarchy is whether the classes P and NP are the same: is P=NP? If a polynomial algorithm were discovered for any NP-complete problem, then all of NP would "collapse" to P; indeed, most of the polynomial hierarchy would disappear.

In algorithmic complexity, two other terms are heard frequently: strongly polynomial and pseudo-polynomial. A *strongly polynomial-time algorithm* is one whose running time is bounded polynomially by a function only of the inherent dimensions of the problem and independent of the sizes of the numerical data. For example, most sorting algorithms are strongly polynomial, because they normally require a number of comparisons polynomial in the number of entries and do not depend on the actual values being sorted; an algorithm for a network problem would be strongly polynomial if its running time depended only on the numbers of nodes and arcs in the network, and not on the sizes of the costs or capacities.

A *pseudo-polynomial-time algorithm* is one that runs in time polynomial in the dimension of the problem and the magnitudes of the data involved (provided these are given as integers), rather than the base-two logarithms of their magnitudes. Such algorithms are technically exponential functions of their input size and are therefore not considered polynomial. Indeed, some NP-complete and NP-hard problems are pseudo-polynomially solvable (sometimes these are called weakly NP-hard or-complete, or NP-complete in the ordinary sense). For example, the integer knapsack problem is NP-Hard, yet it can be solved by a dynamic programming algorithm requiring a number of steps polynomial in the size of the knapsack and the number of items (assuming that all data are scaled to be integers). This algorithm is exponential-time since the input sizes of the objects and knapsack are logarithmic in their magnitudes. However, as Garey and Johnson (1979) observe, "A pseudo-polynomial-time algorithm will display 'exponential behavior' only when confronted with instances containing 'exponentially large' numbers, [which] might be rare for the application we are interested in. If so, this type of algorithm might serve our purposes almost as well as a polynomial time algorithm." The related term strongly NP-complete (or unary NP-complete) refers to those problems that remain NP-complete even if the data are encoded in unary (that is, if the data are "small" relative to the overall input size). Consequently, if a problem is strongly NP-complete then it cannot have a pseudo-polynomial-time algorithm unless P=NP.

Integer Programming

Consider again the problem with which we began the section on linear programming, with the additional constraint that the decision variables must be non-negative integers.

$$\begin{aligned} \text{Maximize:} \quad & z = \sum_{i=1}^{n} v_i x_i \\ \text{Subject to:} \quad & \sum_{i=1}^{n} w_i x_i \le c \\ & x_i \ge 0, \text{and integral}, \; i = 1, 2, 3, \dots, n. \end{aligned}$$

This is the integer knapsack problem. We have already seen that the problem is trivially solved if the x_i are continuous variables: we simply fill the knapsack to capacity with an item that has the highest density $d_k = v_k/w_k$. But can we solve the problem the same way if the x_i are integer values? For example, suppose v_i denotes the Nielsen rating for television program i, and w_i the cost of placing a 30-second spot in program i, for all $i = 1, \dots, n$. Then the knapsack problem asks for a set of commercial insertions that maximize the total gross rating points for a buyer with budget c. For some time media planners thought they could use LP to solve the problem. If the solution gave fractional x_i values, they'd round them off to the nearest integers. That worked well when the budget was large compared to media costs, and when there were few media options. But as the number of options grew, they started getting solutions in which you'd buy 2.3 insertions in program A and 0.8 insertions in program B. With small numbers, it is not clear if rounding off is such a good idea. If you have lots of small numbers of insertions, you could do quite badly. So to make things more realistic (and also a little more interesting), let's restrict the values of x_i to positive integers.

Heuristic and worst-case analysis. We will call the integer knapsack problem P. Suppose we order the items in non-increasing order of density, and then place as many units as possible of the densest item that will fit into the available capacity. How well will this rule do? Not very well. We can show that it can give a solution as bad as 50% of the optimal. That is,(i) it is possible to create a problem in which the optimal solution is twice as large as the solution obtained by selecting items in decreasing density order; and (ii) this is as bad as things can get. This is called worst-case analysis. To show the first claim, we only have to produce an example. To show the second, we need to produce a proof that things cannot get any worse. The two parts together comprise a proof for a tight bound on the performance ratio of an algorithm for a maximization problem.

Let us begin with the first part. We want to show that the density rule can give a solution that is 50% of the optimal. Consider a knapsack problem with two items and (normalized) knapsack capacity $c = 1$. Item 1 has unit weight $w_1 = 1/2$ and unit value $v_1 = 1/2$. Item 2 has unit weight $w_2 = \frac{1}{2} + \epsilon$ and unit value $v_2 = \frac{1}{2} + 2\epsilon$. The optimal solution is to fill the knapsack with 2 units of item 1. As $d_2 > d_1$, the density rule selects one unit of item 2 and stops. The *performance ratio* is

$$r = \frac{\text{value of heuristic solution}}{\text{value of optimal solution}} = \frac{1 \cdot (\frac{1}{2} + 2\epsilon)}{2 \cdot \frac{1}{2}} = \frac{1}{2} + 2\epsilon,$$

which approaches $1/2$ as $\epsilon \to 0$. Now let's show that this is as badly as we can do, this time using a constructive proof.

THEOREM 1. *$r \geq 1/2$ for the density-ordered greedy heuristic.*

PROOF. Consider any knapsack problem with $n \geq 2$ items. Then the item with the highest density occupies at least half the knapsack capacity and so has a solution value no smaller than $dc/2$. The optimal solution can be no larger than dc (why?). Thus,

$$r = \frac{\text{value of heuristic solution}}{\text{value of optimal solution}} \geq \frac{dc/2}{\text{value of optimal solution}} \geq \frac{dc/2}{dc} = \frac{1}{2}.$$

□

The density rule is called the density-ordered greedy heuristic. It is density ordered because it selects items in decreasing density order. It is greedy because it behaves like a greedy child; it has no foresight but picks the immediately most attractive option. And it is a heuristic because it does not always give an optimal solution. Heuristics are also called approximation algorithms. The greedy heuristic is a polynomial-time approximation algorithm because the number of computational steps it requires is polynomial in the problem parameters n, w_i and v_i. We say that it runs in polynomial time. In general, a ρ-approximation algorithm guarantees a performance ratio of at least $0 \leq \rho < 1$ for a maximization problem. As we will see a little later, there is a class of hard problems, called NP-Hard; unless P=NP, there are NP-Hard problems for which $\rho = 0$; others for which ρ is arbitrarily close to 1; and still others for which ρ lies somewhere between the extreme values. The knapsack problem is NP-Hard; we have shown that it admits an approximation algorithm with $\rho = 1/2$.

What is the reason that the density-ordered greedy heuristic does poorly in the worst case? It is that greedy heuristic fills as little space as possible (just over 1/2 of the knapsack capacity). So if all items were small in size (compared to the knapsack capacity), then we would expect the density-ordered greedy heuristic to do better. Let

$$m_i = \lfloor c/w_i \rfloor, \ i = 1, 2, 3, \ldots, n,$$

denotes the maximum number of units of item i that can be placed in the knapsack. Here $\lfloor\cdot\rfloor$ represents "floor" or "rounding to the lower integer." For example $\lfloor 2.1 \rfloor = 2$, $\lfloor 0.9 \rfloor = 0$. Let $m = \min_{1 \leq i \leq n} m_i$. We can generalize the above proof to show that $r \geq m/(m+1)$ for the density ordered greedy heuristic.

Worst-case analysis tells us how bad a heuristic solution can be, compared to the optimal. Obviously we'd like optimal solutions, but if for some reason (which we will discuss shortly) we have to use heuristics, we would like one that does well even in the worst possible cases.

A few years ago, the demand for Dell "Dimensions" started to exceed the company's production (assembly) capacity. One option Dell considered (and subsequently adopted) was to re-design the product for faster assembly, effectively increasing the daily production capacity. The other option was to keep the design unchanged and pay for overtime work. Worst-case analysis can give some insight into when, if at all, Dell should switch from paying overtime to paying for the cost of redesign. To see how, first consider the following consumer decision problem, the tuxedo problem. You are invited to a formal event and need to wear a tuxedo. You can either rent for \$r (e.g., \$50) or purchase for \$$p$ (e.g., \$300) where $p > r$; (we will conveniently assume p to be a multiple of r). So, maybe you decide to rent. After some time, you get invited to another formal event, then another. Now you wish you had bought a tuxedo right at the start.

What is the optimal strategy for renting or buying? The easy situation is when you know far in advance how many times you will need to wear a tuxedo before the style or your waistline changes. What if you don't know ahead of time? What rule can you use? This sort of problem, where you don't know what the future holds, is called an *online problem.* Define the competitive ratio (CR) as the worst case value of the ratio heuristic cost/optimal cost, where "heuristic cost" is the cost for a decision rule (heuristic) and the "optimal cost" (call it "opt") is the lowest cost if you used the best rule *in hindsight.* What is the CR for the rule "buy right away"? If you only go once, you pay \$$p$, but opt is \$$r$. So CR=$p/r$ (e.g, $300/50 = 6$), which can be arbitrarily bad in the worst case. What about a rule that says "rent forever?" If you go a large number of times (e.g., $n = 24$), then the optimal strategy (in hindsight) is to buy and CR=nr/p (e.g., $24 \times 50/300 = 4$), which can also be arbitrarily bad in the worst case.

Is there a better heuristic? Should you wait for some time and then buy even if there is no information in waiting, that is, if the past says nothing about how likely you are to need a tuxedo again? Consider the rule "rent until one more rental would put you at the purchase cost, then buy." (in our example, it is "rent 5 times then buy"). If you end up never buying, then you were optimal. If you buy, you spent $p + (p - r) = 2p - r$ (e.g., $300 + 250$), whereas the optimal in hindsight is to have bought right away for p. So

$$CR = \frac{2p - r}{p} = 2 - \frac{r}{p} \leq 2.$$

So if we use such a rule, we will never pay more than twice the amount we would if we had perfect hindsight. It turns out this is the best possible *deterministic* rent-or-buy rule in terms of the competitive ratio. Can you see why?

Worst-case analysis can be a powerful tool for analyzing consumer decision rules, even dynamic rules under uncertainty. No one in marketing has yet analyzed consumer decision processes in this manner. It is an open area for research. You can use it, for example, to study why and how long people wait to replace products. The furnace[6] in my house is 40 years old; each time it breaks down, I call a mechanic who charges \$c for repairs. I don't know when it will fail next, years may go by or it may happen tomorrow. When should I buy a new furnace, which costs \$d?

Branch and Bound. Let us return to the integer knapsack problem. Is there some algorithm that can solve the problem optimally? There is more than one, including enumeration: calculate the maximum number of units of each item that can be put into the knapsack. Then take all possible combinations of items that can be put into the knapsack. Calculate the solution value for each, and pick the solution with the largest value as the optimal. You can guess why this is not such a good approach. There are 2^n possible subsets of n items (not all subsets constitute a feasible solution). So the number of subsets we need to evaluate increases exponentially with n. Can we do better? One possible method we can

[6]You can replace "furnace" by "car," "television set," or any other consumer durable.

use is called branch and bound; another, which we will consider a little later, is dynamic programming. Both are useful for solving several other types of problems as well, but we will only examine them in the context of the knapsack problem.

Consider the following integer knapsack problem:

$$\begin{array}{ll} \text{Maximize:} & z = 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{Subject to:} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & 0 \leq x_i \leq 1 \text{ and integral, } \ i = 1, 2, 3, \ldots, n \end{array}$$

The LP relaxation gives the solution $x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0, z = 22$. So we know that no integer solution will have value more than 22. We want x_3 to be integer, so we *branch* on x_3, creating two new problems, one with $x_3 = 0$ and the other with $x_3 = 1$; see Figure 1. We again get two non-integer solutions, so we again branch *below the node with the highest non-integer value;* i.e., below $x_3 = 1$, exploring $x_2 = 0$ and $x_2 = 1$ along the two new branches. For $x_2 = 0$, we get the integer solution $x_1 = x_3 = x_4 = 1, x_2 = 0; z = 18$. This value of z is our *lower bound,* and so if we later get any solution value smaller than 18, we know we do not need to branch any further. The other two terminal nodes have solution values $z = 21.65$ and $z = 21.8$; so we choose the latter for branching. Along one branch, we set $x_1 = 0$; along the other, we set $x_1 = 1$. The latter is infeasible; the former gives the solution $x_1 = 0, x_2 = x_3 = x_4 = 1; z = 21$. This is an integer solution; its objective function value is as large as the *upper bound* $\lfloor 21.65 \rfloor = 21$ if $x_3 = 0$, and so we know we have an optimal solution.

To summarize:

1. Solve the linear relaxation of the problem. If the solution is integer, then we are done. Otherwise create two new subproblems by branching on a fractional variable.
2. A subproblem is not active when any of the following occurs:
   (a) you used the subproblem to branch on;
   (b) all variables in the solution are integer;
   (c) the subproblem is infeasible;
   (d) you can fathom the subproblem by a bounding argument.
3. Choose an active subproblem and branch on a fractional variable. Repeat until there are no active subproblems.

Depending on the type of problem, the branching rule may change: if x is restricted to be integer (but not necessarily 0 or 1), then if $x = 4.27$ you would branch with the constraints $x \leq 4$ and $x \geq 5$ (not on $x = 4$ and $x = 5$). Branch-and-bound can be used for any integer programming problem. In the worst case, the number of subproblems can be so large that we in effect have to enumerate all solutions.

Dynamic Programming. We now examine another way to solve the integer knapsack problem — this time an algorithm that produces the optimal solution without enumerating all solutions.

Suppose we have *integer* weights w_i and *integer* knapsack capacity c. This is equivalent to assuming that the weights and the capacity are rational numbers (why?) We will construct a matrix F with n rows and c columns. The i-th row corresponds to the i-th item in some arbitrary numbering of the n items; the j-th column corresponds to a knapsack with capacity j, where $1 \leq j \leq c$. The ij-th entry, denoted $f(i, j)$, gives the value of the optimal solution if we fill a knapsack of capacity j with (some) subset of the items $1, 2, 3, \ldots, j$; $f(i, j) = 0$ if there is no such subset of items. Suppose we select x_i units of item $i, 0 \leq x_i \leq m_i$, where m_i is the maximum number of units of item i that can fit into capacity j. Then

$$f(i, j) = \max\{f(i-1, \quad j - x_i w_i) + x_i v_i \mid 0 \leq x_i \leq m_i\},$$

where the maximum is taken over the feasible integer values of x_i. For example, suppose $m_i = 1$. Then either item i is placed in the knapsack or it is not. If it is, then

$$f(i, j) = f(i-1, \quad j - w_i) + v_i.$$

If it is not, then

$$f(i, j) = f(i-1, \ j).$$

Thus,

$$f(i, j) = \max\ \{f(i-1, j), \quad f(i-1, \quad j - w_i) + v_i\}.$$

This is a recursion; we can use it to fill all elements of F and obtain the optimal solution value $F(n, c)$. First fill out the first row of F :

$$f(1, j) = v_1, j < w_1;$$
$$f(1, j) = 2v_1, w_1 \leq j < 2w_1;$$
$$\vdots$$
$$f(1, j) = m_1 v_1, j \geq m w_1.$$

Next, fill out the second row using the recursion, and so on until the table is finished. The table has nc entries. Suppose each entry takes a constant time to compute; we can fill out the table in time proportional to nc. We also do not have to keep the entire matrix: only the previous row and the new one being filled, so the space requirement is proportional to c, the knapsack capacity.

We have just constructed a dynamic programming algorithm for the integer knapsack problem. Why is it a dynamic program? Because to write row i, we only need to know the values in rows $i-1$. All previous rows can be discarded. This is the key property of a dynamic program — the states of a current stage depend only on the values of the states in the previous stage. By a "stage," I mean a row of the matrix F. By a "state," I mean a column of the matrix. More formally, we call the *item number* in an ordered sequence a *stage variable* and the *unused capacity* at a stage a *state* variable.

Let us try to prove that the dynamic program gives the optimal solution to the integer knapsack problem. It seems obvious, but how do we prove it? It is possible to construct a proof by contradiction (try it). But let us use another method, called mathematical induction, which works only if the relevant variables take finite, integer values. We will prove our result by induction over n, the number of items, which appear as stages in the dynamic program.

THEOREM 2. *The dynamic program solution for the integer knapsack problem is optimal.*

PROOF. We prove the result by induction on n, the number of items.

Base case: $n = 1$. If we have an integer knapsack problem with a single item, then the optimal solution is to fill in as many units of the knapsack as possible; it is trivial to verify that the dynamic program selects this solution, because

$$f(1,j) = \max\{f(0,\ j - x_1w_1) + x_1v_1 \mid 0 \le x_1 \le m_1\} = \max\{x_1v_1 \mid 0 \le x_1 \le m_1\} = m_1v_1.$$

Induction hypothesis: Suppose

$$f(i,j) = \max\{f(i-1,\ \ j - x_iw_i) + x_iv_i \mid 0 \le x_i \le m_i\},\ \text{for all } 1 \le i \le n-1.$$

Induction step: To prove that

$$f(n,c) = \max\{f(n-1,\ \ c - x_nw_n) + x_nv_n \mid 0 \le x_n \le m_n\}.$$

Suppose the optimal solution has a value of z^* and that it contains $0 \le x_n^* \le m_n$ units of item n. Let $c^{'} = c - w_nx_n^*$. Consider the knapsack problem

$$\begin{aligned} \max z^{'} &= \sum_{i=1}^{n-1} v_ix_i \\ \text{s.t.}\qquad & \sum_{i=1}^{n-1} w_ix_i \le c^{'} \\ & x_i \ge 0 \text{ and integal} \end{aligned}$$

Then

$$z^* = x_n^*v_n + z^{'},$$

where $z^{'} \geq 0$ denotes the contribution to the optimal solution value from the remaining items, $i = 1, 2, 3, \ldots, n-1$, which together occupy at most $c^{'}$ of the knapsack capacity. From the induction hypothesis

$$z^{'} = f(n-1, c^{'}),$$

and so

$$\begin{aligned} z^* &= x_n^* v_n + f(n-1, c^{'}), \\ &\leq \max\,\{f(n-1, \ \ c^{'}) + x_n v_n \mid 0 \leq x_n \leq m_n\} \\ &= f(n, c). \end{aligned}$$

But z^* is the optimal solution, and so $z^* \geq f(n, c)$. It follows that

$$z^* = f(n, c),$$

which is the desired result. $\square$

We seem to have a satisfactory way of dealing with the integrality aspect of the media planning problem. Let's go back and introduce some of the other constraints we had done before. First, let us suppose that the objective function is not linear. Suppose instead it were concave, reflecting diminishing returns to insertions, or s-shaped, reflecting both minimum thresholds and diminishing returns. Can you accommodate these in the dynamic program? Next, let us add upper and lower bounds of the type

$$b_i \leq x_i \leq c_i, \ i = 1, 2, 3, \ldots, n.$$

How these change the solution to P_2? Finally, let us add constraints of the type

$$x_i + x_j \leq c_{ij}.$$

Can you amend the dynamic program to reflect these constraints?

Some Marketing Applications of Integer Programming

Google Maps. To generate driving directions, online services like Google maps and Mapquest first transmit the starting and destination addresses entered into the browser window to an application server that locates these points on a road-network database. This process is called geocoding. Rather than storing individual street addresses, road databases are organized into road segments — one side of a single block, for instance. Each segment is represented by a string of 256 characters that contains its name and address information, latitude and longitude, and other important attributes such as road class, speed, turn and links to other connecting segments. A typical U.S. road database contains eight million to 10 million road segments and tens of thousands of "points of interest" — airports, museums, businesses and so forth. One road database occupies several gigabytes of memory. Once the addresses have been matched to road segments (149 Main Street would be located midway on the 101-199 Main Street segment), a shortest-path algorithm is used to calculate an "optimal route" between the segments.[7] The software translates the resulting set of connecting road segments into a narrative that the user can understand, like "Merge onto Bruckner Expressway in 2.7 miles."

[7]This description is taken from "Atlas Shrugged," p. 20, of the November 2000 issue of *Scientific American.*

Bellman-Ford algorithm. This is a dynamic-programming algorithm for the shortest-path problem. It is similar to the above DP algorithm for the knapsack problem. Let $a_{ij} \geq 0$ denote the (possibly infinite) length of the edge (arc) connecting vertices (nodes)i and j. Let d_j denote the length of the shortest path from the origin to vertex j. We number the origin 1, and the other vertices $2, \ldots, n$. Set $d_1 = 0$. For each vertex $j \neq 1$, there must be some final edge (k, j) in a shortest path from 1 to j. Let $d_{j|k}$ denote the length of the shortest path that goes to (vertex) 1 to j via k. Then

$$d_{j|k} = d_k + a_{kj}.$$

This is the "Principle of Optimality." It says that $d_{j|k}$ is obtained by adding the length a_{kj} to the length of the shortest path from 1 to k. Now k can only correspond to the vertices $k = 1, \ldots, j-1, j+1, \ldots, n$. We therefore should select k as a vertex for which $d_{j|k}$ is as small as possible. We say that the shortest-path lengths must satisfy the following Bellman equations:

$$d_1 = 0, \quad d_j = \min_k d_{j|k}, \quad \text{for all } j = 2, \ldots, n.$$

So far, we have argued that this condition is necessary for a shortest path from 1 to n. Can you show that it is also sufficient?[8] Can you use the Bellman equations to solve the shortest-path problem using a dynamic program similar to the one above for the knapsack problem?

Dijkstra's algorithm. This is a greedy algorithm for the shortest-path problem. It can be used if, as we assume above, all the edge lengths are non-negative. Start with the destination vertex, n. Let $j(\neq n)$ denote a vertex for which

$$a_{jn} \leq a_{in} \quad \text{for all} i \neq j.$$

That is, j is the closest neighbor of n. Let $d_j^* = a_{jn}$. Set

$$a_{in} = \min\{a_{in}, a_{ij} + a_{jn}\} \quad \text{for all vertices } i \neq j, n.$$

Remove vertex j from the graph. Apply the same steps to the new graph. Stop when you have computed the value of d_1^* (or more generally, when there are no more vertices left to remove). Can you prove that this algorithm obtains the optimal solution? How would the algorithm change if you start with vertex 1 instead of vertex n?

Retail merchandise testing (Fisher and Rajaram 2000). In a merchandise depth test, a retail chain introduces new products at a small sample of selected stores for a short time before the primary selling season and uses the observed sales to forecast demand for the entire chain. Fisher and Rajaram (2000) describe a method for resolving two questions in merchandise testing: (1) which stores to use for the tests and (2) how to extrapolate from test sales to create a forecast of total season demand for each product in the chain. The procedure requires the retailer to have a record of the sales history of comparable products offered in one or more prior seasons.

[8] We have assumed throughout that the lengths are all non-negative. More generally, it is possible to have negative lengths. The Bellman-Ford algorithm provides a method for checking if there are any negative length cycles; see, for example, Bertsimas and Tsitsiklis, op. cit..

Define a *primary sales season* as the period within a sales season during which (1) the selling price of a product exceeds acquisition cost plus variable selling expenses, and (2) the inventory at each store is sufficient to prevent supply shortages. Prices can vary over time during the test period, but are assumed to be the same at any time in all test stores for the duration of the test. This precludes the noise introduced by substitution sales. One also wants the set of products available during the testing to be as similar as possible to those that will be available during the regular season.

Let n denote the number of stores in the chain and m the number of previous products with an available sales history. Let S_{ip} denote the observed sales during the primary sales season at store i for product p. Let $S = \sum_i S_{ip}$. Let $\bar{S}_{ip}$ denote the sales of product p in store i during a period comparable in timing and duration to a period during which a test is to be conducted.

Store selection. To choose k test stores, we partition the n stores of the chain into k clusters. The stores within each cluster are chosen to minimize a measure of dissimilarity based on the percentage of total sales represented by sales of each of the prior products in each store. Thus, two stores selling exactly the same percentage of each of the prior products will be in the same cluster. We then choose a single test store within each cluster that best represents the cluster, in the sense that using test sales at this store to forecast sales of other stores in the cluster minimizes the cost of forecast errors. Let I denote the set of stores and let P denote the set of prior products. Let

$$y_j = \begin{cases} 1 & \text{if store } j \text{ is chosen as a test store;} \\ 0 & \text{otherwise;} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if store } i \text{ is assigned to a cluster represented by test store } j; \\ 0 & \text{otherwise;} \end{cases}$$

$$w_i = \sum_{p \in P} S_{ip}, \quad \beta_{ip} = \frac{S_{ip}}{w_i}, \ i \in I, p \in P,$$

$$d_{ij} = \sum_{p \in P} (U_p \max(\beta_{ip} - \beta_{jp'}, 0) + O_p \max(\beta_{jp} - \beta_{ip'}, 0)),$$

where U_p is the unit cost of understocking (e.g., unit margin) and O_p is the unit cost of overstocking (e.g., unit cost - final discuntes sales price) item $p \in P$. The test store selection problem (TSSP) is given by the following integer program:

$$\min \sum_{i \in I} \sum_{j \in I} w_i d_{ij} x_{ij}$$

subject to:

$$\sum_{j \in I} x_{ij} = 1, \quad i \in I$$

$$\sum_{j \in I} y_j = k$$

$$0 \leq x_{ij} \leq y_j \leq 1, \quad i, j \in I$$

$$x_{ij}, y_j = 0, 1 \text{ integral}, \quad i, j \in I.$$

The first constraint ensures that each each store is assigned to a single test store. The second constraint requires that we have exactly k test stores. The third constraint restricts store assignments to the chosen test stores. The objective function minimizes the total overstock and understock costs if test sales at each test store are extrapolated to develop forecasts for stores assigned to it. Thus, the solution to the integer program gives a clustering of stores and select a test store within each cluster.

REMARK. The above problem has the structure of a *k-median problem*: given n points in a metric space, we must select k points as cluster centers, and then assign each point to its closest selected center. If point j is assigned to a center i, the cost incurred is proportional to the distance between i and j. The goal is to select the k centers that minimize the sum of the assignment costs. The problem is NP-Hard, but there are well-known heuristics. The first constant-factor approximation algorithm was proposed by Tardos (1998); it has a worst-case performance bound of 7. There are recent claims of a 3-approximation algorithm. For earlier work, see Cornuejols, Fisher and Nemhauser (1977).

Scheduling Television Commercials. Clients approach NBC with a plan request. Those who usually pay high premiums get top priority. Sales planners work on the sales requests in order of priority, developing a detained schedule of commercial slots to be assigned to a client that meets both the client's and the network's requirements. The most important criterion is to ensure that the plan does not give more high-premium inventory to a client than necessary to win the contract. The client specifies the budget, show mix, weekly weighting and unit mix. NBC management ranks the shows and weeks in a year by their importance. These, together with the available inventory, are used to specify weight R_{sw} that measure the value of inventory on show s in week w.

The sales-planning problem is to minimize the amount of premium inventory assigned to a plan, subject to constraints on inventory (availability and conflict) and client requirements. The decision variables are the number of commercials of each spot length the client requests in the shows and weeks included in the sales plan.

Let Q denote the set of quarters. Let W_q^Q denote the set of all weeks in quarter q. Let W denote the set of all weeks. Let S denote the set of shows. Let L =set of allowable commercial lengths requested by a client. The lengths are expressed as multiples of 30-second units. Let x_{swl} denote the number of commercials of length ℓ on show s in week w assigned to the plan, $s \in S, w \in W$. Let P_{sw} denote the rate-card price of a 30-second spot on show s in week w.

There are two budget constraints. The first is:

$$B_1 \leq \sum_{s\in S}\sum_{w\in W}\sum_{\ell\in L} lP_{sw}x_{swl} \leq B_2$$

which says that the total cost must lie within the lower and upper bound on the total plan budget (in rate card dollars). The second budget constraint is:

$$B_{q1} - \sigma_{q1}^B \leq \sum_{s\in S}\sum_{w\in W_q}\sum_{\ell\in L} \ell P_{sw}x_{swl} \leq B_{q2} + \sigma_{q2}^B,\ q \in Q$$

which says that the total cost in a quarter must lie within a lower and upper bound of the budget; the σ terms are used to introduce penalties in the objective function for violations of the quarterly budget range, rendering this a soft constraints.

The show-mix constraints are:

$$\alpha_{s1}u - \sigma^S_{s1} \leq \sum_{w\in W}\sum_{\ell\in L} \ell x_{swl} \leq \alpha_{s2}u + \sigma^S_{s2}, s \in S,$$

where the α terms denote the bounds on the fraction of commercials to be aired on show s; and the σ terms denote the associated slack.

The total audience for the plan is

$$\sum_{s\in S}\sum_{w\in W}\sum_{\ell\in L} lA_{sw}x_{swl}$$

where A_{sw} are the number of people in the client's demographic expected to watch show s in week w. Then the constraint

$$\beta_{w1}g_q - \sigma^W_{w1} \leq \sum_{s\in S}\sum_{\ell\in L} \ell A_{sw}x_{swl} \leq \beta_{w2}g_q - \sigma^W_{w2}, w \in W_q, q \in Q$$

says that the total audience must be a weighted sum of the quarterly audiences

$$g_q = \sum_{s\in S}\sum_{w\in W_q}\sum_{\ell\in L} lA_{sw}x_{swl}$$

where weights are the β terms denoting the bounds on the fraction of total impressions in the quarter realized in week w. The σ terms denote the associated slacks for week w.

The unit-mix constraints are:

$$\gamma_{\ell 1}u + \sigma^U_{\ell 1} \leq \sum_{s\in S}\sum_{w\in W} \ell x_{swl} \leq \gamma_{\ell 2}u + \sigma^U_{\ell 2},\ \ell \in L$$

where

$$u = \sum_{s\in S}\sum_{w\in W}\sum_{\ell\in L} \ell x_{swl}$$

is the total 30-second equivalent units assigned to the plan; the γ terms refer to the bounds on the fraction of total units in the plan to be of length ℓ; and the σ terms are the slacks on the fraction of total units on the plan to be of length ℓ.

The air-time inventory constraint is:

$$\sum_{\ell\in L} \ell x_{swl} \leq I_{sw}, s \in S, w \in W,$$

where the right hand side is the number of 30-second equivalent spots available on show s in week w. Finally, the product-conflict constraint is:

$$\sum_{\ell\in L} x_{swl} \leq x_{sw}, s \in S, w \in W,$$

where the right hand side is the maximum number of commercials that can be aired on show s in week w for the client without violating product-conflict constraints.

The objective function is:

$$\text{Minimize} \sum_{s \in S} \sum_{w \in W} \sum_{\ell \in L} \ell R_{sw} x_{swl} + \sum_{i=1}^{2} \left(\sum_{q \in Q} \pi^B \sigma_{qi}^B + \sum_{s \in S} \pi^S \sigma_{si}^S + \sum_{w \in W} \pi^W \sigma_{wi}^W + \sum_{\ell \in L} \pi^U \sigma_{\ell i}^U \right),$$

where the π terms are the linear penalties associated with violating the bounds on the (soft) constraints.

An LP-relaxation, followed by rounding down is used to generate an initial, feasible solution. The variables are then sorted in descending order of their fractional parts in the LP solution. Each variable in that order is considered for incrementing up to the next integer value. If it is feasible, and if the objective function value increases, its value is changed; otherwise, the next variable is considered. After all the variables are considered, an improved "intial" solution is attained. Next, a decision is made whether to add or delete a single unit in a show airing included in the plan. The plan is then re-evaluated. The process stops after a fixed number of steps. The data entry time is 5-10 minutes, the algorithm running time is typically 2 minutes. The total time for generating a sales plan is 15-20 minutes, down from the earlier 3-5 hours. A separate, ISCI rotator is used to assign specific commercials to programs at a later date if the buyer has purchased hundreds of time slots in a broadcast season. The constraint in the placement is that two separate airings of a particular execution (item) have to be placed as far apart as possible. The inputs are the available slots; the number of different commercials to be aired in these slots; and the number of times each commercial has to be aired. The problem is NP-Hard. A greedy algorithm is used for the assignment. First, sort the data in ascending order of n_j, the number of insertions of commercial j. Start with an item with highest n_j; if there are N slots, assign to every q_jth slot where $q_j = \lfloor N/n_j \rfloor$ if the fractional part of q_j is no larger than 1/2 and to $q_j = \lfloor N/n_j \rfloor + 1$ if the fractional part of q_j exceeds 1/2. If this is infeasible, find the closest neighbor at each step. Repeat for each item in the list. Stop when all items are assigned.

Conjoint-Based Optimal Product Design. A conjoint problem with m product attributes has $\prod_k n_k$ possible product profiles, $1 \leq k \leq m$. Some of these are more attractive to a firm than others: they might get higher market share, or cost less to make. How do we determine which product concepts are the most attractive? If there is one consumer or segment, the answer is simple; but if there are many consumers with different preferences over the attributes, then it is harder to tell which product concepts offer the most potential for further development. Note that we do not want a single "optimal" product profile at this stage: we want a number of potential candidate profiles, which are feasible, and which are not trivially dominating (the best of all features at the lowest price).

Zufryden (1977) formulated the problem of optimal product design using conjoint data. Kohli and Krishnamurti (1987) described an alternative graph-theoretic formulation of the problem, showed it was NP-Hard and described three heuristics for solving the problem. Kohli and Sukumar (1991) extended the formulation to product-line design, a problem that was earlier examined in a different way by (Green and Krieger (1985). Heuristics using other methods (e.g., beam search by Thakur, Nair and Wen (1993), genetic algorithms by Balakrishnan and Varghese (1996))) have since been reported for the problems.

We begin by describing the problem of designing a single optimal product and then consider the product-line design problem. We consider both the deterministic choice rule used in the above noted papers and probabilistic generalizations of the rules.

Optimal design of a single product. Consider a directed graph $G(V, E)$. Let $s \in E$ denote a source node and let $t \in E$ denote a terminal node. In addition, each level of each attribute is represented by a node in the graph. For notational convenience, we label $j(k)$ the node associated with level j of attribute k, $1 \leq j \leq n_k, 1 \leq k \leq m$. Then s has no inbound edges and n_1 outbound edges, terminating into each of the n_1 nodes $j(1)$ associated with the levels of attribute 1; each node $j(1)$ has n_2 outbound edges, terminating into each of the n_2 nodes $j(2)$ associated with the levels of attribute 2. In general, a node $j(k)$ has n_{k-1} incoming nodes from each of the nodes representing the levels of attribute $k-1$, and n_{k+1} outbound nodes into the nodes $j(k+1)$ representing the levels of attribute $k+1$, $1 \leq k \leq m-1$. The destination node has no outbound edges, and n_m inbound arcs from nodes representing the levels of attribute m.

It is trivial to see that each $s-t$ path in G corresponds to a product profile. Let u_{ijk} denote individual i's part worth for level j of attribute k (if we use a main-effects conjoint model). Then we associate for each individual i a weight $w_i[j(k)] = u_{ijk}$ with node $j(k)$, $1 \leq i \leq n$, $1 \leq j \leq n_k$, $1 \leq k \leq m$. For a conjoint model including (selected) second-order interaction effects, we associate a weight $w_i[j(k), j(k+1)]$ with the arc connecting the nodes $j(k)$ and $j(k+1)$, $1 \leq j(k) \leq n_k$, $1 \leq j(k+1) \leq n_{k+1}$, $1 \leq k \leq m-1$, $1 \leq i \leq n$. We can then allow second-oder interaction terms between pairs of successive attributes represented in the graph. For simplicity, we will discuss the main-effects model below; the extension to the interaction-effects case is trivial.

Each consumer i has a status-quo product, represented by a sequence of nodes (attribute levels)

$$\{j_i^*(k) \mid 1 \leq k \leq m\}, 1 \leq i \leq n.$$

For each individual and attribute level, we compute the normalized weights within individual and attribute:

$$v_i[j(k)] = w_i[j(k)] - w_i[j_i^*(k)].$$

Now consider an $s-t$ path in G with the weights $v_i[j(k)]$ associated with the nodes. If the sum of the weights along the path exceeds zero, then person i prefers the associated product profile to the status-quo profile; otherwise, s/he prefers the status quo profile.

The *share of choices* (SOC) problem is to find a (product profile) that maximize the number (share) of individuals who prefer the selected profile to their status quo. Note that this is a deterministic utility formulation; we have entirely ignored the error in parameter estimates.

The SOC problem corresponds to finding a $s-t$ path in G for which the sum of weights $v_i[j(k)]$ exceeds zero for the maximum number of individual. We can show that this problem is NP-Hard as follows. Consider a SOC problem in which each attribute has two levels and all $v_i[j(k)]$ values are 0 or 1. Then a product profile preferred by individual i to status quo corresponds to a $s-t$ path along which the sum of weights is non-zero. The decision version of the problem is: is there an $s-t$ path for which the sum of weights exceeds zero for at least r individuals, where r is any integer $1 \leq r \leq r$. We can formulate the problem as a $0-1$ integer program by using the graph structure. the problem is in NP because we can count in a polynomial number of steps the number of individuals for whom the sum of weights along a given $s-t$ path exceeds zero. We can show that it is NP-Complete by demonstrating that this special case of the SOC problem corresponds to the following problem.

Satisfiability. Given n clauses, each a disjunction of m truth (true/false) variables, is there a truth assignment that satisfies at least r clauses?

The proof comprises showing that each clause can be represented by an individual in a SOC problem, and each truth variable can be represented by a (binary) attribute; see Kohli and Krishnamurti (1986) for details.[9] In light of this result, we examine heuristic solution procedures. One method is to perform a Lagrangian relaxation of the integer program; it performs poorly because the relaxation is quite weak. A second method is to construct a dynamic-programming heuristic in which the attributes are stages and the attribute levels are states. One then progresses from s to t in the graph G by moving along the stages (attributes) and keeping for each state (attribute level) that *partial* product profile leading up to it that maximizes the SOC for the considered subset of attributes. To illustrate the main ideas, consider a small example with three binary attributes and three individuals (practical problems in general have ten or more attributes at many levels each and hundreds of individuals). We represent the values of $v_i[j(k)]$ by the following matrices:

$$\begin{array}{c c} i & \begin{array}{cc} V_1(1) & V_2(1) \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \left(\begin{array}{cc} -1 & 2 \\ 1 & 2 \\ -2 & 2 \end{array}\right) \end{array}$$
with $V(1) =$ the matrix above;

$$V(2) = \begin{array}{c c} i & \begin{array}{cc} V_1(2) & V_2(2) \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \left(\begin{array}{cc} 3 & -1 \\ -3 & -4 \\ 1 & 4 \end{array}\right) \end{array}$$

$$V(3) = \begin{array}{c c} i & \begin{array}{cc} V_1(3) & V_2(3) \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \left(\begin{array}{cc} 3 & -1 \\ 3 & -1 \\ 3 & -1 \end{array}\right) \end{array}$$

where row i of column $V_j(k)$ contains individual i's normalized part worths for level j of attribute k.

Step 1: Construct the matrices

$$C[1(2)] = V(1) + [V_1(2) \;\; V_1(2)]$$
$$C[2(2)] = V(1) + [V_2(2) \;\; V_2(2)]$$

[9]It is worth noting that the satisfiability representation corresponds to the case when a person's decision for selecting an item is based on a disjunctive rule; i.e., on a rule that classifies a product profile preferable to the status quo if it is better than the latter on at least one attribute.

Here,

$$C[1(2)] = \begin{pmatrix} -1 & 2 \\ 1 & 2 \\ -2 & 2 \end{pmatrix} + \begin{pmatrix} 3 & 3 \\ -3 & -3 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 5 \\ -2 & -1 \\ -1 & 3 \end{pmatrix}$$

$$C[2(2)] = \begin{pmatrix} -1 & 2 \\ 1 & 2 \\ -2 & 2 \end{pmatrix} + \begin{pmatrix} -1 & -1 \\ -4 & -4 \\ 4 & 4 \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ -3 & -2 \\ 2 & 6 \end{pmatrix}$$

The first column of both $C[1(2)]$ and $C[2(2)]$ has one positive element and the second column has two column elements; we pick the second column from each matrix and arrange these to form the matrix $C(k)$ for $k = 2$:

$$C(2) = \begin{pmatrix} 5 & 1 \\ -1 & -2 \\ 3 & 6 \end{pmatrix}$$

The first (second) column of $C(2)$ contains the revised weights associated with the node representing level 1 (level 2) of attribute 2; the second column of $C(2)$ contains the revised weights associated with the node representing level 1 of attribute 2.

Step 2. Construct the matrices

$$C[1(3)] = C(2) + [V_1(3) \;\; V_1(3)],$$
$$C[2(3)] = C(2) + [V_2(3) \;\; V_2(3)].$$

Here

$$C[1(3)] = \begin{pmatrix} 5 & 1 \\ -1 & -2 \\ 3 & 6 \end{pmatrix} + \begin{pmatrix} 3 & 3 \\ 3 & 3 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 8 & 4 \\ 2 & 1 \\ 6 & 9 \end{pmatrix}$$

$$C[2(3)] = \begin{pmatrix} 5 & 1 \\ -1 & -2 \\ 3 & 6 \end{pmatrix} + \begin{pmatrix} -1 & -1 \\ -1 & -1 \\ -1 & -1 \end{pmatrix} = \begin{pmatrix} 4 & 0 \\ -2 & -3 \\ 2 & 5 \end{pmatrix}$$

Both columns of $C[1(3)]$ contain three positive elements; we select column 1 because it "dominates" column 2 in each row (otherwise, we break ties by first considering the number of non-negative elements, then the sum of column elements). We also select column 1 of $C[2(3)]$ because it contains two positive elements, whereas column 2 of the matrix contains one positive element. We use these columns to form the matrix

$$C(3) = \begin{pmatrix} 8 & 4 \\ 2 & -2 \\ 6 & 2 \end{pmatrix}$$

Step 3: We terminate the algorithm at this stage because we have $C(k)$ for $k = 3$; i.e., we have considered all three attributes. The solution produced by the DP heuristic is contained in the column of $C(3)$ with the largest number of positive elements (if necessary, ties are broken as before). In this case, the solution is given by the first column of $C(3)$ which has all three positive elements. The corresponding product profile (obtained by keeping a list of selected levels for each attribute) has level 1 of each attribute. There are additional details for dealing with infeasible solutions (by carrying multiple columns). In practical settings, one wants to rule these out in an interactive setting, because it can be very difficult to specify all infeasible solutions ahead of time.

Although the problem has since been solved using a variety of different heuristics - genetic algorithms, beam search, simulated annealing — none does any better in the worst case. From an empirical standpoint, the DP does quite well, as do each of the heuristics proposed afterwards.

There are several issues with using the SOC model in practice. First, it assumes deterministic preferences. Second, it does not include any measure of uncertainty in market share estimates. Third, it maximizes market share, not profit or consumer welfare. Fourth, it does not account for the cannibalization of a new product's introduction. Fifth, it assumes that a single product is to be introduced.

The problem of assuming deterministic preferences has not been addressed in the literature. It is important because deterministic share estimates can be biased. It is obviously useful to have a measure of uncertainty reported along with the share estimates; it is better still if one could somehow reflect the uncertainty in the maximization itself, by modifying the objective function. We will consider this below. The use of market share as a response measure is driven by our inability to measure multi-attribute cost functions, which are central to estimating a profit function. Variable costs are easier to associate with attribute levels, but fixed costs, which can vary substantially depending on the configuration of an existing product line and the features of a selected product, are much harder to estimate. As far as measures of consumer welfare are concerned, there have been efforts to build models that maximize the sum of utilities across individuals. But this has the limitation that it does not consider the distribution of utilities across consumers and it requires interpersonal utility comparisons, which is not a reasonable demand from conjoint data. Gupta and Kohli (1989) discuss the issue in greater detail and suggest alternative formulations of the consumer welfare maximization problem. Cannibalization effects can be assessed by modifying the objective function so that individuals for whom the status-quo corresponds to a product made by the firm developing a new product are eliminated from the optimization. Finally, the method can be extended to configure optimal product lines. We discuss this below.

Probabilistic choice. Suppose we estimate a conjoint model using regression (in which case the error term is normal) or logit (in which case it is logistic). Let π_{ip} denote the probability that a test profile p is selected from a choice set that also contains one or more currently favored (status quo) items. Note that because of a probabilistic choice rule, we can no longer restrict attention to a single status quo item with the highest utility. Instead, we need to include all items a consumer either currently buys or may consider for purchase.[10] Let S denote the set status quo items. We change the dynamic programming heuristic as follows: at each step $1 \leq k \leq m$, we compute for each individual the purchase probability π_{ijk} of a partial profile terminating in state (level) j of attribute k, $1 \leq j \leq n_k$. The relation between π_{ijk} and the utilities associated with the partial product profiles is specified

[10]Models of consideration sets can also be introduced; see, e.g., Kohli and Jedidi 2002).

by the error distribution of the utility function. In the case of a multinomial logit model, it has the form

$$\pi_{ijk} = \frac{e^{u_{ijk}}}{e^{u_{ijk}} + \sum_{s \in S} e^{u_{ik}(s)}} = \frac{1}{1 + \sum_{s \in S} e^{-(u_{ijk} - u_{ik}(s))}}$$

where $u_{ik}(s)$ denotes individual i's utility for the partial profile of product $s \in S$ in which we consider only attributes $1, 2, 3, \ldots, k$. The exponents of the summation terms in the denominator are the relative utilities, except that these are now computed for all status quo items, not just for the item with the highest utility.

Next, we compute the associated weighted mean purchase probability[11]

$$\bar{\pi}_{jk} = \frac{1}{nr} \sum_{i=1}^{n} r_i \pi_{ijk}.$$

where r_i denote the purchase rate per time unit for individual $i, 1 \leq i \leq n$, and

$$r = \sum_{i=1}^{n} r_i$$

denotes the total unit sales during a time unit in the category. Estimates of r_i can be obtained either by directly asking consumers, or by asking about the frequency of category purchases and then transforming the latter into relative purchase rates by the method described by Ehrenberg (19xx). The algorithm then proceeds as before, except that the criterion for selecting a partial profile terminating into level j of attribute k is the maximum value of the mean purchase probability obtained by augmenting the n_{k-1} partial product profiles terminating into the levels of attribute $k-1$ by level j of attribute k, $1 \leq k \leq m$. The expressions for the probabilities change in an obvious manner if one uses a multinomial probit model instead of multinomial logit.

Details of a deterministic method extension of the above method for constructing product lines are given in detail by Kohli and Sukumar (1991). Here, we consider a different approach, due to Green and Krieger (1985), for product line design.

Product line design. We begin with an $n \times t$ matrix U of individual-by-profile utilities

$$U = \begin{array}{c} i \\ 1 \\ 2 \\ 3 \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} u_1 & u_2 & u_3 & \ldots & u_t \end{array} \\ \begin{pmatrix} u_{11} & u_{12} & u_{13} & \ldots & u_{1t} \\ u_{21} & u_{22} & u_{23} & \ldots & u_{2t} \\ u_{31} & u_{32} & u_{33} & \ldots & u_{3t} \\ & & \vdots & & \\ u_{n1} & u_{n2} & u_{n3} & \ldots & u_{nt} \end{pmatrix} \end{array}$$

Column u_j contains the utilities (relative to a staus quo) the n individuals associate with product profile j, $1 \leq j \leq t$. Without loss of generality, we assume that one of the columns of U is all zeroes; it corresponds to a status quo item for each

[11]It is also possible to use information about higher order moments of the empirical distribution of the purchase probability of partial profiles.

individual. Let

$$V = \begin{array}{c} i \\ 1 \\ 2 \\ 3 \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & \dots & v_t \end{array} \\ \begin{pmatrix} v_{11} & v_{12} & v_{13} & \dots & v_{1t} \\ v_{21} & v_{22} & v_{23} & \dots & v_{2t} \\ v_{31} & v_{32} & v_{33} & \dots & v_{3t} \\ & & \vdots & & \\ u_{n1} & u_{n2} & u_{n3} & \dots & u_{nt} \end{pmatrix} \end{array}$$

denote a $n \times t$ matrix of seller's returns if consumer i chooses product j.

Let $T = \{j \mid 1 \leq t \leq m\}$ denote the index set of product profiles and let $N = \{i \mid 1 \leq i \leq n\}$ denote the index set of consumers. Green and Krieger (1985) assume a "first choice" rule: each consumer $i \in N$ selects a product profile $j_i \in \tau$ that has the maximum utility:

$$u_{ij_i} = \max_{j \in \tau} u_{ij}$$

Their *seller's problem* is to find a subset of ℓ items so that the seller's return is maximized. Let $\mathbf{T}$ denote the set of all ℓ item subsets; it has $\binom{t}{\ell}$ elements. Then we can write the seller's problem as

$$\max_{\tau \in \mathbf{T}} z = \sum_{i=1}^{n} v_{ij_i}$$
$$\text{s.t.} \quad u_{ij_i} = \max_{j \in \tau} \; u_{ij}, \; 1 \leq i \leq n,$$

The constraint says that each person selects an item with the highest utility. Note that if a person selects a status quo item, the seller's return can be zero (if the status quo item is offered by a competitor) or positive (if it is offered by the firm). We can formulate this problem as the k-median problem, which is NP-Hard, but for which good heuristics are known (Cornejeouls, Fisher and Nemhauser 1977). The simplest

is a greedy heuristic, which incrementally selects product profiles to maximize the marginal return to a seller; its worst-case bound is $1 - e^{-1}$. It runs as follows:

(1) *Initialization (step 1).* Let $v_i^* = v_{ij}$ if $u_{ij} > 0$. Compute $v = \sum_i v_i^*$. Select column j_1 for which v is maximum.

(2) *Recursion (step s).* Select a second candidate column, j. Update $v_i^* = v_{ij}$ if $u_{is} > \max\{0, u_{ij_1}, u_{ij_2}, u_{ij_3}, \ldots, u_{ij_{s-1}}\}$. Otherwise, keep v_i^* unchanged. Compute $v = \sum_i v_i^*$. Select column j_s for which v is maximum.

(3) *Termination:* Stop if $s = \ell$.

Green and Krieger also describe a "buyers" problem, which is equivalent to setting $V = U$. However, as noted earlier, this approach to designing products for consumer welfare has several problems.

Call planning (Lodish 1977). A salesperson's territory is divided into sales calling units (SCUs), each sufficiently small that can be served by a single salesperson. An SCU may be a small town, or a collection of villages close together, or a contiguous part of a large city. We assume that once a salesperson visits an SCU, s/he calls on multiple accounts, then returns to a home office.

Let x_i be an integer variable denoting the number of calls a salesperson makes on account i, $1 \le i \le n$. Let $0 \le \underline{x}_i \le x_i \le \bar{x}_i$, where $\underline{x}_i$ and $\bar{x}_i$ denote the smallest and largest number of calls allowed on account i during a planning period. Let g_i denote the geographic area in which account i is located, $1 \le g_i \le m$. Suppose that if a salesman is in the area in which account i is located, then each call on account i takes t_i time units; we assume that $t_i > 0$ is an integer value, $1 \le i \le n$; i.e., the call time for account i is an integer multiple of the time unit, which can be as small small as we like. Let e denote the number of *effort periods* during a planning horizon; and let $s_j \ge 0$ denote the number of trips to area j during an average effort period, $1 \le j \le m$. Let $r_i(x_i)$ denote the sales response from account i if x_i calls are made over a planning period; let a_i denote an account-specific "adjustment" factor, which we will discuss below; let c_i denote the cost of making a trip to area j; and let t_j denote the travel time to area j. Finally, let t denote the total time available to a salesperson over a planning period. Then the optimal calling plan is the solution to the following integer programming problem P :

$$\text{Maximize:} \qquad z = \sum_{i=1}^{n} a_i r_i(x_i) - e \sum_{j=1}^{m} s_j c_j$$

$$\text{Subject to:} \qquad \sum_{i=1}^{n} t_i x_i + \sum_{j=1}^{m} s_j u_j \le t$$

$$0 \le \underline{x}_i \le x_i \le \bar{x}_i \text{ and integer, } 1 \le i \le n.$$

Let

$$r_i(x_i) = r_{i0} + \frac{\Delta r_i}{1 + k x_i^{\alpha}},$$

where r_{i0} denotes the sales response with zero calls; $r_{i0} + \Delta r_i$ denotes the maximum possible sales response; and $\alpha < 0$ is a response parameter. Thus, if one has three additional data points besides r_0, one can estimate k and α. The points often used are the current call effort, and $\pm 50\%$ call efforts. The response values for these points, and the a_i values, are obtained from judgment by a salesperson; the optimization uses the values of the function at integer values of x_i, so we still have an integer programming problem to solve.

We discussed several ways of solving knapsack problems in the last chapter. We can use these methods if there were some way to reduce P to the form of a knapsack problem. Lodish (1977) achieves this by the following two-step approach. First, restrict attention to accounts in a single area, $j,\ 1 \leq j \leq m$. Let P_1 denote the problem

$$\begin{aligned} &\text{Maximize:} \qquad && z = \sum_{i=1}^{n} a_i r_i(x_i) - e s_j c_j \\ &\text{Subject to:} && \sum_{i=1}^{n} t_i x_i + s_j u_j \leq \tau \\ & && 0 \leq \underline{x}_i \leq x_i \leq \bar{x}_i \text{ and integer, } 1 \leq i \leq n. \end{aligned}$$

Let $\tau_j = 0, 1, 2, \ldots, t$, denote the maximum time spent in area j. We replace τ on the right hand side above by a fixed value τ_j, and solve the problem for each fixed value $1 \leq s_j \leq s^*$. Call the resulting problem P_1 :

$$\begin{aligned} &\text{Maximize:} \qquad && z_j(\tau_j, s_j) = \sum_{i=1}^{n} a_i r_i(x_i) - e s_j c_j \\ &\text{Subject to:} && \sum_{i=1}^{n} t_i x_i \leq \tau_j - s_j u_j \\ & && 0 \leq \underline{x}_i \leq x_i \leq \bar{x}_i \text{ and integer, } 1 \leq i \leq n. \end{aligned}$$

This is a knapsack problem because $e s_j c_j$ and $s_j u_j$ are constants. For each value of $\tau_j = 0, 1, 2, \ldots, t$, we compute

$$z(\tau_j) = \max_{s_j} z_j(\tau_j, s_j), \ \ \tau_j = 0, 1, 2, 3, \ldots, t.$$

Finally, we solve another knapsack problem P_2 :

$$\begin{aligned} &\text{Maximize:} \qquad && z = \sum_{j=1}^{m} z_j(\tau_j) \\ &\text{Subject to:} && \sum_{i=1}^{n} \tau_j \leq t. \end{aligned}$$

Lodish (1977) describes several successful applications of this model, called CALLPLAN. It is an example of an early decision support system; after it came a series of extensions to territory alignment/design and sales force sizing, which also saw substantial commercial use. We will examine one such problem below. However, instead of extending CALLPLAN for territory alignment decisions (which is possible, and has been done by Lodish), we will consider a different and also successful approach, described by Zoltners and Sinha (1983).

Territory alignment (Zoltners and Sinha 1983). Consider the problem of combining n SCUs into m sales territories, where $m < n$; typically, $m \ll n$. Let $a_j,\ 1 \leq j \leq n$, denote an attribute of an SCU; e.g., workload, sales potential, sales volume. Let d_{ij} denotes the distance between the center of SCU j and the center of territory i. Hess and Samuels (1971) consider the model

$$\begin{aligned}
\text{Minimize:} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} a_j d_{ij}^2 x_{ij} \\
\text{Subject to:} \quad & \sum_{j=1}^{n} a_j x_{ij} = \frac{1}{m}\sum_{j=1}^{n} a_j,\ 1 \leq i \leq m, \\
& \sum_{i=1}^{m} x_{ij} = 1,\ 1 \leq j \leq n, \\
& x_{ij} = 0, 1(\text{integral}),\ 1 \leq i \leq m, 1 \leq j \leq n.
\end{aligned}$$

The objective function, which minimizes the sum of weighted squared distances, is interpreted as maximizing the compactness of a sales territory; the weights are the SCU scores on a relevant attribute. The first constraint restricts the assignment of an SCU to one territory. The second constraint requires that the territories be "balanced" on an attribute (e.g., workload or sales potential). A solution to the integer program is a "compact" sales territory that equalizes a single attribute among territories. Unfortunately, the model is often over-constrained because no feasible solution exists that uniquely assign each SCU to a territory and also guarantees equal territory scores on an attribute. If the integrality constraint is relaxed, the resulting linear program is a transportation model and can be readily solved. Fortunately, at most $m-1$ SCUs will be split across territories; Hess and Samuels suggest rounding off the fractional values, and find that in seven applications, the rounded solutions created territories within $\pm 10\%$ of balance. Note that the sales territories produced by the model need not consist of contiguous or connected SCUs. There is also an emphasis on a single attribute. Segal and Weinberger (1977) incorporate accessibility into the model by replacing $a_j d_{ij}^2$ by the distance of the shortest path between SCU j and the center of territory i via the network connecting adjacent, traversable SCUs. Sales territories designed with this objective function tend to accommodate travel lanes and geographic obstacles like mountains and waterways.

Zoltners (1976) extends Hess and Samuel's model to accommodate multiple attribute balancing:

$$\begin{aligned}
\text{Minimize:} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{Subject to:} \quad & l_{ik} \leq \sum_{j=1}^{n} a_{ijk} x_{ij} \leq u_{ik},\ 1 \leq i \leq m,\ 1 \leq k \leq h, \\
& \sum_{i=1}^{m} x_{ij} = 1,\ 1 \leq j \leq n, \\
& x_{ij} = 0, 1(\text{integral}),\ 1 \leq i \leq m, 1 \leq j \leq n.
\end{aligned}$$

Here, c_{ij} is the distance associated with the assignment of SCU j to territory i; k is an index for the h relevant territory attributes; a_{ijk} is the value of the kth attribute if SCU j is assigned territory i; and l_{ik} (u_{ik}) is the lower (upper) bound on attribute k for territory i. Relaxing the integrality constraint on the x_{ij} and solving the problem using linear programming produces a solution with at most m SCUs that are split. We can then use branch-and-bound to solve the problem.

Zoltners and Sinha (1983) describe the following important method for territory alignment using a hierarchical SCU-adjacency tree. An adjacency tree is a graph whose vertex is the SCU containing the territory center. Nodes of the graph represent other SCUs which are candidates for inclusion in the sales territory. Edges of the graph connect SCUs which are adjacent via a feasible road network. Figures 1 show a geographic region with 17 SCUs; Figure 2 overlays a road network on the region; and Figure 3 shows the asociated *road graph*.[12]

The first step is to find the path with the shortest travel time between each territory center and the surrounding SCUs. This can be done (for any graph) using the following greedy algorithm due to Dijkstra. Start with a set A containing only a (given) territory center. Let all other SCU's belong to a set B. At each step, move from B to A a node with the smallest distance from s. Stop when all nodes are in A.[13]

[12]Zoltners and Sinha report that they have developed a computer database of all U.S. interstate highways, all U.S., highways and all state highways.

[13]The shortest path between a pair of nodes also contains the shortest path for any pair of intermediate nodes on the shortest path.

This produces the shortest-path adjacency tree (see Figure 4) from which the shortest-path SCU-adjacency tree (Figure 5) is obtained by placing the territory center at the root of the tree. SCUs that are immediately adjacent to the territory center are the first adjacency levels; SCUs which are one level removed are at the second level, and so forth. Assuming that shortest paths will be used to represent distances between territory centers and SCUs, the following contiguity property can be stated in terms of the shortest path SCU-adjacency tree: if an SCU at the kth level is assigned to a territory center, sales territory contiguity is ensured when the preceding SCU at the $(k-1)$st level is also assigned to the territory center. Zoltners and Sinha use this property in their model.

Let A_{ij} denote the set of SCUs that immediately precede SCU j on any branch of the hierarchical SCU-adjacency tree for center i, $,1 \leq i \leq m, 1 \leq j \leq n$. Let a_{ijk} denote the value of the kth attribute for SCU j, conditioned on the assignment of SCU j to territory center $i, 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq h$. Zoltners and Sinha propose the following integer program for determining the alignment of sales territories:

$$\text{Minimize:} \qquad \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ijk^*} x_{ij}$$

$$\text{Subject to:} \qquad l_{ik} \leq \sum_{j=1}^{n} a_{ijk} x_{ij} \leq u_{ik},\ 1 \leq i \leq m,\ 1 \leq k \leq h, k \neq k^*,$$

$$x_{ij} \leq \sum_{p \in A_{ij}} x_{ip},\ 1 \leq i \leq m,\ 1 \leq j \leq n,$$

$$\sum_{i=1}^{m} x_{ij} = 1,\ 1 \leq j \leq n,$$

$$x_{ij} = 0, 1(\text{integral}),\ 1 \leq i \leq m, 1 \leq j \leq n.$$

Lagrangian Relaxation. Consider the following single-attribute, weighted distance model:

$$\text{Minimize:} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} a_j d_{ij} x_{ij}$$

$$\text{Subject to:} \quad \sum_{j=1}^{n} a_j x_{ij} = b_i, \ 1 \le i \le m,$$

$$x_{ij} \le \sum_{p \in A_{ij}} x_{ip}, \ 1 \le i \le m, \ 1 \le j \le n$$

$$\sum_{i=1}^{m} x_{ij} = 1, \ 1 \le j \le n$$

$$x_{ij} = 0, 1(\text{integral}), \ 1 \le i \le m, 1 \le j \le n$$

where b_i is the balancing term (e.g., $b_i = \sum_{j=1}^{n} a_j/m$). First, drop the constraints

$$x_{ij} \le \sum_{p \in A_{ij}} x_{ip}, \ 1 \le i \le m, \ 1 \le j \le n.$$

A Lagrangian relaxation of the problem is obtained by dropping the constraints

$$\sum_{j=1}^{n} a_j x_{ij} = b_i, \ 1 \le i \le m,$$

and adding the term

$$\sum_{i=1}^{m} \lambda_i (\sum_{j=1}^{n} a_j x_{ij} - b_i)$$

to the objective function:

$$\text{Minimize:} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} a_j (d_{ij} - \lambda_i) x_{ij} + \sum_{i=1}^{m} \lambda_i b_i$$

$$\text{Subject to:} \quad \sum_{i=1}^{m} x_{ij} = 1, \ 1 \le j \le n$$

$$x_{ij} = 0, 1(\text{integral}), \ 1 \le i \le m, 1 \le j \le n.$$

Reversing the order of summation in the first term of the objective function gives

$$\text{Minimize:} \sum_{j=1}^{n}\sum_{i=1}^{m} a_j (d_{ij} - \lambda_i) x_{ij} + \sum_{i=1}^{m} \lambda_i b_i$$

which can be written as

$$\text{Minimize:} \sum_{j=1}^{n} a_j \Big[\sum_{i=1}^{m} (d_{ij} - \lambda_i) x_{ij} + \lambda_i b_i \Big]$$

The solution to the relaxation is thus trivial:

$$x_{ij} = 1 \text{ if } d_{ij} - \lambda_i = \min_p \{d_{pj} - \lambda_p\}; \text{otherwise, } x_{ij} = 0.$$

We can prove (e.g., by contradiction) that this solution satisfies the contiguity conditions, which we had earlier dropped from the problem. It now remains to find the values of λ_i for which the relaxation takes its upper bound; i.e., for which it achieves the highest value.[14] Zoltners and Sinha report that the territory assignments obtained using this method are usually within 5% of complete balance. A similar alignment procedure is used in the general case where there are multiple attributes. The contiguity property need not hold any longer, but is enforced by assigning each noncontiguous SCU in the Lagrangian-relaxation solution to their closest contiguous center.

Exercises

1. A marketing research company wishes to conduct a mail survey to investigate the growing interest in the sport of indoor rock climbing. To do this, it will buy mailing lists from various magazines. The company wants to reach both males and females already involved in some athletic or outdoor activity. The cost for a mailing list from the leading rock climbing magazine is \$200 and will give a mailing list of 110,00 readers. Of these readers 75% are male and 25% are female. The leading men's fitness magazine reaches 200,000 readers, 10% of which are women. The cost for this mailing list is \$500. The leading women's fitness magazine charges \$400 for their mailing list and reaches 175,000 readers. Of these readers, 85% are women. The leading general outdoor recreation magazine has a readership of 250,000, of which 50% are women and 50% are men. The mailing list for this magazine will cost \$350. The leading mail order outdoor outfitter firm sells the mailing list of its catalog members for \$100 and reaches 50,000 people, of which 40% are women and 60% men. The marketing firm has allocated \$1000 for purchase of these mailing lists. The company feels that the larger part of those who are becoming indoor rock climbers are men and

[14] A sub-gradient method (e.g., Held, Wolfe and Crowder 1974) can be used for this purpose.

therefore would like to reach at least twice as many men as women. Determine which the most appropriate mailing lists for the marketing firm to purchase.

2. Paint Fair Co. advertises its weekly sales in newspapers, television, and radio. Each dollar spent in advertising in newspapers is estimated to reach 12 buying customers; each dollar in TV reaches 15 buying customers; and each dollar in radio reaches 10 buying customers. The company will spend at least 20% of its ad budget of $17,000 in each medium. The combined newspaper and TV budget will not be larger than three times the radio budget. How much should the company spend on each medium if it is interested in reaching as many buying customers as possible?

3. A company wishes to plan an advertising campaign in television, radio, and magazines. The purpose of the advertising program is to reach as many potential customers as possible. The result of a market study are given below:

	TV prime time	TV day time	Radio	Magazines
Cost of an advertising unit	$40,000	75,000	30,000	15,000
# potential customers reached/unit	400,000	900,000	500,000	200,000
# women customers reached/unit	300,000	400,000	200,000	100,000

The company does not want to spend more than $800,000 on advertising. It further requires that (i) at least 2 million exposures take place among women; (ii) advertising on TV be limited to $500,000; (iii) at least 3 advertising units be bought on day time TV, and two units during prime time; and (iv) the number of advertising units on radio and magazines should each be between 5 and 10.

4. Glass Discount Store is opening a new department with a storage area of 10,000 sq.ft. Management considers four products for display. Below is the cost c_i, selling price p_i and storage requirement s_i for product i, $i = 1, 2, 3, 4$.

$c_1 = \$55, \quad p_1 = \$130, s_1 = 24$ sq.ft./unit
$c_2 = \$100, \quad p_1 = \$120, s_1 = 20$ sq.ft./unit
$c_3 = \$55, \quad p_1 = \$295, s_1 = 36$ sq.ft./unit
$c_4 = \$300, \quad p_1 = \$399, s_1 = 50$ sq.ft./unit

At least 10 units of each product must be on display. How many units of each product should be on display if the company has $600,000 available for purchasing the products and if the company's objective is to maximize profits?

(a) You are in charge of an advertising campaign for a new product, with a budget of $1 million. You can advertise on TV or in magazines. One minute of TV costs $20,000 and reaches 1.8 million potential customers; a magazine page costs $10,000 and reaches 1 million. You must sign up for at least 10 minutes of TV time. How should you spend your budget to maximize your audience?
(b) It takes creative talent to create effective advertising; in your organization it takes three person-weeks to create a magazine page, and one person-week

to create a TV minute. You have only 100 person-weeks available. Add this constraint to the model and determine how you should spend your budget.

(c) Radio advertising reaches a quarter million people per minute, costs \$2,000 per minute and requires only 1 person-day of time. How does this medium affect your solutions?

(d) How does the solution change if you have to sign up for at least two magazine pages? A maximum of 120 minutes of radio?

5. Find the equilibrium strategies in a two-person, zero sum game for the following payoff matrix

$$A = \begin{pmatrix} 0.1 & -0.2 & 0 & -0.05 \\ -0.2 & 0.15 & -0.1 & 0.05 \\ 0.05 & -0.05 & -0.2 & 0.1 \end{pmatrix}$$

6. Show that $r \geq m/(m+1)$ for the density ordered greedy heuristic for the integer knapsack problem.

7. Formulate and interpret the dual of the linear program for MSN's banner-advertising problem.

8. What is the relationship between the ordinal regression model using LINMAP and the above two regression models?

9. What are the conditions under which you will recommend the use of OLS and each of the above two methods for estimating a regression model?

10. Remember when people bought LP records — those vinyl disks with songs on both sides? The Beatles recorded 7 songs for one album (it might have been *Abbey Road,* but I am not sure). The duration of each song in minutes is:

Song:	1	2	3	4	5	6	7
Time:	2	5	2	2	7	2	2

Which songs should appear on the side containing no more than half the total music time so that there are as many songs as possible on this "short" side of the album?

11. Using the dynamic programming, find the optimal solution to an integer knapsack problem with knapsack capacity $c = 12$ and five items with the following weights and values:

i	w_i	v_i
1	8	80
2	6	48
3	2	14
4	3	18
5	4	22

(a) Compare the solution to that obtained using the density-ordered greedy heuristic.

(b) Are there any other greedy heuristics you could use?
(c) Can you construct a greedy heuristic that has a better worst-case bound than the density-ordered greedy heuristic? (Note: this is a hard question).

12. Let $U = \{u_1, u_2, u_3, \ldots, u_m\}$ be a set of Boolean *variables.* A *truth assignment* for U is a function $t : U \rightarrow \{T, F\}$. If $t(u) = T$ we say that u is "true" under t; if $t(u) = F$ we say that u is "false." If u is a variable in U, then u and $\bar{u}$ are *literals* over U. The literal u is true under t if and only if the variable u is true under t; the literal $\bar{u}$ is true if and only if the variable u is false. A *clause* over U is a set of literals over U, such as $\{u_1, \bar{u}_3, u_8\}$. It represents the disjunction of those literals and is *satisfied* by a truth assignment if and only if at least one of its members is true under that assignment. The clause above will be satisfied by t *unless* $t(u_1) = F, t(u_3) = T$ and $t(u_8) = F$. A collection of clauses over U is *satisfiable* if and only if there exists some truth assignment for U that simultaneously satisfies all the clauses in C. Such an assignment is called a *satisfying truth assignment.* The SATISFIABILTY problem is specified as follows:

SATISFIABILITY (SAT)

INSTANCE: A set U of variables and a collection C of clauses over U.

QUESTION: Is there a satisfying truth assignment for C?

For example, $U = \{u_1, u_2\}$ and $C = \big\{\{u_1, \bar{u}_2\}, \{\bar{u}_1, u_2\}\big\}$ is an instance of SAT for which the answer is "yes." A satisfying truth assignment is given by $t(u_1) = t(u_2) = T$. On the other hand, replacing C by $C^{'} = \big\{\{u_1, \bar{u}_2\}, \{\bar{u}_1, u_2\}, \{\bar{u}_1\}\big\}$ yields an instance for which the answer is "no"; $C^{'}$ is not satisfiable.

(a) What is the relation between SAT and the problem of inferring a *disjunctive screening rule,* using which a person judges an alternative (e.g., brand) as acceptable if it satisfactory on at least one relevant attribute? (Hint: consider the special case in which each of a set of alternatives is described using binary attributes.)
(b) What is the relation between SAT and a *conjunctive screening rule,* using which a person judges an alternative as acceptable if it is satisfactory on all relevant attributes? (Hint: consider the relation between disjunctive and conjunctive rules.)
(c) SAT is NP-Complete. What does this imply about the difficulty a person (or a computer) faces when inferring disjunctive and conjunctive screening rules from data?
(d) Suppose we have $0-1$ data indicating which alternatives in a set are acceptable to a person and which are not acceptable. Each alternative is described over m binary attributes.
   (i) Formulate the problem of inferring a conjunctive classification of alternatives as an integer program.
   (ii) Describe a heuristic for solving the problem (which is NP-hard).
   (iii) *Extra credit:* Derive the worst-case bound for the heuristic.
   (iii) *More extra credit:* Show that a maximization version of satisfiability can be represented as a special case of the share-of-choice problem in optimal product design.

13. Remember the solution to the "tuxedo problem?" Suppose you want to test if people actually use such a rule when making decisions under uncertainty.
    (a) Identify five other situation in which people might use a similar decision strategy.
    (b) Specify three to five hypotheses to test if people do in fact use the kind of decision process suggested by the tuxedo problem.
    (c) Describe competing hypotheses regarding how people might make these decisions.
    (d) Design one or more experiment to test the proposed and competing hypotheses. Be specific in terms of the variables you will manipulate, the tasks you will ask subjects to perform, the measurements you will obtain and the analyses you will do to confirm or disconfirm the hypotheses.
14. **Traveling Salesman Problem.** Consider a sales territory with n towns or cities. A tour consists of a route through each of the n cities starting with and concluding at a salesman's base city. The traveling salesman problem (TSP) is to identify the lowest-cost tour of the cities, given a cost (in dollar or time units) c_{ij} of going from city i to city j, $1 \leq i < j \leq n$. The symmetric version of the problem imposes the condition $c_{ij} = c + ji$; the asymmetric version imposes no such condition. Describe a greedy heuristic for solving the problem. Prove that the heuristic does not obtain the optimal solution. (Both versions of TSP are NP-hard, even if the "costs" are Euclidean distances satisfying the triangular inequality: $c_{ij} \leq c_{ik} + c_{kj}$ for any triple i, j, k of cities. Papadimitriou and Vempala (1999) show that the asymmetric traveling salesman problem cannot be approximated to an approximation ratio better than 41/40; the symmetric version of the problem cannot be approximated to better than 129/128.)
15. Show that the Bellman optimality condition for the shortest-path problem is necessary and sufficient for solving the shortest-path problem. Use the condition to construct a dynamic-programming algorithm for solving the shortest-path problem.
16. The ijth element of the following matrix gives the expected time in minutes to drive along road segments directly connecting locations i and j on a map.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} & 20 & 90 & \infty \\ & & 80 & 70 \\ & & & 90 \end{pmatrix} \end{array}$$

    For example, locations 1 and 2 are connected by a road segment of length 2; and there is no road segment that directly connects locations 1 and 4. Suppose a person using Google Maps asks for driving directions from location 1 to location 4. Use Dijkstra's algorithm to give the person directions that minimize the expected driving time. How will the solution change if a traffic accident causes a traffic delay of 20 minutes on the segment connecting locations 1 and 2?
17. Show that the dynamic-programming heuristic for the optimal product design problem can perform arbitrarily badly in the worst case.
18. Modify the probabilistic dynamic-programming heuristic to construct a product line of ℓ items so that the mean market share for the product line is maximized.
19. Formulate the seller's problem when consumers i chooses product p with probability $\pi_i(p|s)$ from a choice set s, $1 \leq i \leq n$. Modify the above greedy heuristic

to allow a probabilistic choice rule (e.g., logit) for consumers when selecting an alternative from an offered product line.

20. Aside from the use of a deterministic choice rule, what are the limitations of Green and Krieger's formulation of the seller's product-line design problem? Suggest how one can overcome at least one of these limitations. Be specific.