Revised Simplex Method

1. Theory of the Simplex Method

- **Standard form.** All constraints must be equalities and all variables are non-negative.
- **BFS.** Setting the *n*-*m* variables equal to 0, and the remaining m variables have the unique nonnegative values.
- **Adjacent BFSs.** For a LP with *m* constraints (not including sign restrictions), two BFSs are **adjacent** to each other if they share *m*-1 basic variables.

Theorem 2.2 The feasible region for any LP has only a finite number of extreme points.
Theorem 2.3 Any LP that has an optimal solution has an extreme point that is optimal.

Theorem 4.1 An extreme point in the feasible region of an LP is a bfs to the LP, and vice versa.

Property 1: There are only a *finite* number of BFSs.
Property 2 (Theorem 4.2): (*a*) If there is exactly one optimal solution, then it must be a BFS. (*b*) If there are multiple optimal solutions (and a bounded feasible region), then at least two must be adjacent BFSs.
Property 3: If a BFS has no *adjacent* BFSs that are *better* (as measured by *Z*), then there are no *better* BFSs anywhere. Therefore, such a BFS is guaranteed to be an *optimal* solution (by Property 2), assuming only that the problem possesses at least one optimal solution (guaranteed if the problem possesses feasible solutions and a bounded feasible region).

2. Prepare for the simplex method: Canonical form

A problem with this structure is said to be in **canonical form**.

1. All decision variables are constrained to be nonnegative.
2. All constraints, except for the nonnegativity of decision variables, are stated as equalities.
3. The right-hand-side (rhs) coefficients are all nonnegative.
4. One decision variable is isolated in each constraint with a +1 coefficient. The variable isolated in a given constraint does not appear in any other constraint.

In order to transform a general LP problem to canonical form of the augmented system, it is convenient to perform the necessary transformations according to the following sequence:

1. Replace each decision variable unconstrained in sign by a difference between two nonnegative variables. This replacement applies to all equations including the objective function.
2. Multiply the constraints with a negative rhs coefficient by −1.
3. Change inequalities to equalities by the introduction of slack and surplus variables. For >=inequalities, let the nonnegative *surplus (excess) variable* represent the amount by which the left-hand side exceeds the rhs; for <=inequalities, let the nonnegative *slack variable* represent the amount by which the rhs exceeds the left-hand side (lhs).
4. Add a (nonnegative) *artificial variable* to any equation that does not have an isolated variable readily apparent, and construct the phase I objective function or modify the original objective function.

Any feasible solution to the augmented system with all artificial variables equal to zero provides a feasible solution to the original problem. Conversely, every feasible solution to the original problem provides a feasible solution to the augmented system by setting all artificial variables to zero.

3. Sketch of Simplex Method: an iterative algorithm

Matrix format: $\max \quad z = \mathbf{cx}$

$s.t., \quad \mathbf{Ax} = \mathbf{b}$

$\mathbf{x} \geq \mathbf{0}$

Step 1 &2 Initialization:
Set up to start iterations, including constructing the canonical form and finding an initial BFS.

Iteration:
Step 3(a) Optimality test: Is the current BFS solution optimal? If Yes, Stop; otherwise, perform an iteration to find (move to) a better adjacent BFS.

Optimality condition: A canonical form is optimal (for a max problem) if each nonbasic variable has a nonpositive coefficient in **revised objective function (rof)**.Note that the coefficients for basic variables in rof are always 0 because their effect on z is counted in the constant in rof.

$$\textbf{rof: } z = \mathbf{c}_{BV}(\mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_{NBV}) + \mathbf{c}_{NBV}\mathbf{x}_{NBV} = \mathbf{c}_{BV}\mathbf{B}^{-1}\mathbf{b} + (\mathbf{c}_{NBV} - \mathbf{c}_{BV}\mathbf{B}^{-1}\mathbf{N})\mathbf{x}_{NBV}$$

Step 3(b). Choose an entering variable based on rof

Step 4. Choose a leaving variable using minimum ratio test

$$\mathbf{B}^{-1}\mathbf{B}\mathbf{x}_{BV} + \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_{NBV} = \mathbf{B}^{-1}\mathbf{b} \quad \Rightarrow \mathbf{x}_{BV} + \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_{NBV} = \mathbf{B}^{-1}\mathbf{b}$$

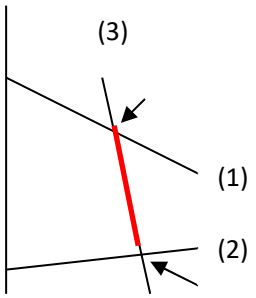
Step 5. Replace the leaving basic variable with the entering variable and generate new adjacent BFS by updating B and **inverse of B** (revised simplex method).

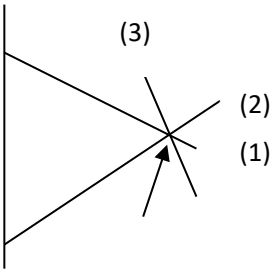
Three special cases

a. **Infeasible LP: determined in phase I or big-M method**
b. **Multiple optimal solutions:** If there is no nonbasic variable with a zero coefficient in the optimal rof, the LP has a unique optimal solution. If there is a nonbasic variable with a zero coefficient in the optimal rof, it is possible that the LP may have alternative optimal solutions.
c. **Unbounded:** an unbounded LP for a max problem occurs when a variable with a positive coefficient (maximization problem) in rof has a non-positive coefficient in each constraint – the ratio test fails!

Alternative optimal solutions:

Even if there is a nonbasic variable with a zero coefficient in the optimal rof, it is possible that the LP may not have alternative optimal solutions. The first figure shows that two extreme points are optimal and all points in the line segment are optimal. The second figure shows that two sets of basic variables are optimal, but result in the same optimal solution.





4. Revised Simplex Method

The revised simplex method is based directly on the matrix form of the simplex method. However, the difference is that the revised simplex method incorporates a key improvement into the matrix form. Instead of needing to invert the new basis matrix **B** after each iteration, which is computationally expensive for large matrices, the revised simplex method uses a much more efficient procedure to simply updates $\mathbf{B}^{-1}$ from one iteration to the next. Next, we describe this procedure.

Let

x_k = entering basic variable;

a'_{ik} = revised coefficient of x_k in current row (i), for $i = 1,2,\dots,m$, i.e., revised coefficient column vector $(a'_{1k}, a'_{2k}, \dots, a'_{mk})^{-1}$ corresponding to x_k in $\mathbf{B}_{\text{old}}^{-1}$;

r = index of row containing the leaving basic variable.

All that is needed to update $\mathbf{B}^{-1}$ from one iteration to the next is to obtain the new $\mathbf{B}^{-1}$ (denoted it by $\mathbf{B}_{\text{new}}^{-1}$) from the old $\mathbf{B}^{-1}$ (denoted it by $\mathbf{B}_{\text{old}}^{-1}$) by performing the usual algebraic operations on $\mathbf{B}_{\text{old}}^{-1}$ (on the entire system of equations except the rof) for this iteration (i.e., Gaussian elimination). That is,

$$\mathbf{E}\mathbf{a}'_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & & -\frac{a'_{1k}}{a'_{rk}} & 0 & & 0 \\ 0 & 1 & & -\frac{a'_{2k}}{a'_{rk}} & 0 & & 0 \\ \vdots & 0 & \cdots & \vdots & & \cdots & \vdots \\ \vdots & \vdots & & \frac{1}{a'_{rk}} & 1 & & \vdots \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & & -\frac{a'_{mk}}{a'_{rk}} & 0 & & 1 \end{bmatrix} \begin{bmatrix} a'_{1k} \\ a'_{2k} \\ \vdots \\ a'_{rk} \\ a'_{(r+1)k} \\ \vdots \\ a'_{mk} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

This procedure can be expressed in terms of matrix as

$$\mathbf{B}_{\text{new}}^{-1} = \mathbf{E}\mathbf{B}_{\text{old}}^{-1}$$

where matrix $\mathbf{E} = [\mathbf{U}_1, \mathbf{U}_2, \dots, \mathbf{U}_{r-1}, \boldsymbol{\eta}, \mathbf{U}_{r+1}, \dots, \mathbf{U}_m]$. The m elements of column vector $\mathbf{U}_i$ are 0 expect for a 1 in the ith position. That is, matrix $\mathbf{E}$ is an identify matrix except that its rth column is replaced by the vector

$$\boldsymbol{\eta} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}, \text{ where } \eta_m = \begin{cases} -\frac{a'_{ik}}{a'_{rk}} & if\ i \neq r \\ \frac{1}{a'_{rk}} & if\ i = r. \end{cases}$$

The revised simplex method consists of combining this procedure for updating $\mathbf{B}^{-1}$ at each iteration with the rest of the matrix form of the simplex method illustrated in class. One advantage of the revised simplex method (matrix form) is that the number of arithmetic computations may be reduced. This is especially true when the **A** matrix contains a large number of zero elements (which is usually the case for the large problems arising in practice). The amount of information that must be stored at each iteration is less, sometimes considerably so. The revised simplex method also permits the control of the round-off errors inevitably generated by computers. This control can be exercised by periodically obtaining the current $\mathbf{B}^{-1}$ by directly inverting **B**. Furthermore, some of the postoptimality analysis (sensitivity analysis) problems can be handled more conveniently with the revised simplex method. For all these reasons, the revised simplex method in matrix form is usually the preferable form of the simplex method for computer execution.

5. Two-Phase Method (developed from Big-M method)

Artificial variables are added to place the LP in canonical form. Phase I objective function, denoted as w, equals the sum of all the artificial variables. Minimizing w will result in one of the three cases.

Case 1 The optimal value of w is greater than zero. Then, the original LP is **infeasible**.
Case 2 The optimal value of w is equal to zero, and no artificial variables are in the optimal Phase I basis. In this case, we drop all columns in the optimal Phase I that corresponds to the artificial variables (i.e., setting artificial variables to zero).
Case 3 The optimal value of w is equal to zero and at least one artificial variable is in the optimal Phase I basis. In this case, at the end of Phase I we drop from the optimal Phase I all nonbasic artificial variables and can also drop any variable from the original problem that has a positive coefficient in the optimal Phase I rof (i.e., nonbasic variable).

We then combine the original objective function with the constraints from the optimal Phase I. This yields the **Phase II LP.** Phase II starts with the BFS found in Phase I. The optimal solution to the Phase II LP is the optimal solution to the original LP.

Thinking: *How to identify the constraints that cause the infeasibility of a problem?*

6. Degeneracy and Convergence

- Theoretically, the simplex algorithm can fail to find the optimal solution to an LP.
- LPs arising from actual applications seldom exhibit this unpleasant behavior.
- Consider the following relationship for a Max LP:

  z-value for new BFS = z-value for current BFS+ (value of entering variable in new BFS) × (coefficient of entering variable in rof of current BFS)

  Recall: (coefficient of entering variable in rof) > 0 and (value of entering variable in new BFS) >= 0.

  Two Facts:
(1) If (value of entering variable in new BFS) >0, then (z-value for new BFS) > (z-value for current BFS);
(2) If (value of entering variable in new BFS) =0, then (z-value for new BFS) = (z-value for current BFS).

- An LP is nondegenerate if in each of the LP's BFSs, all of the basic variables are positive.

  For a nondegenerate LP, Fact 1 implies that each iteration of the simplex will *strictly increase z*, and consequently, it is impossible to encounter the same BFS twice. So for solving a nondegenerate LP, the simplex algorithm is guaranteed to find the optimal solution in a finite number of iterations.

- Any BFS that has at least one basic variable equal to zero is a **degenerate BFS**.
- **An LP is degenerate** if it has at least one degenerate BFS.
- For an LP with n decision variables to be degenerate, (n+1) or more of the LP's constraints (including the sign restrictions as constraints) must be binding at an extreme point. Such an extreme point corresponds to more than one set of basic variables. Such an extreme point corresponds to a **degenerate BFS.** That is, more than one set of basic variables may generate the same BFS, and, consequently, correspond to a given extreme point. If this is the case, then the LP is **degenerate**.

- **Bland's rule** showed that cycling can be avoided by applying the following rules (assume that slack and excess variables are numbered $x_{n+1}, x_{n+2}, \ldots$):
  1. Choose as the entering variable (in a max problem) the variable with a positive coefficient in rof that has the smallest subscript.
  2. (Tie breaking for the leaving basic variable) If there is a tie in the ratio test, then break the tie by choosing the winner of the ratio test so that the variable leaving the basis has the smallest subscript.

7. The interior-point approach to solving linear programming problems

One meaningful way of comparing interior-point algorithms with the simplex method is to examine their theoretical properties regarding computational complexity. Karmarkar has proved that the original version of his algorithm is a **polynomial time algorithm;** i.e., the time required to solve *any* linear programming problem can be bounded above by a polynomial function of the size of the problem. Pathological counterexamples have been constructed to demonstrate that the simplex method does not possess this property, so itis an **exponential time algorithm** (i.e., the required time can be bounded above only by an exponential function of the problem size). This difference in *worst-case performance* is noteworthy. However, it tells us nothing about their comparison in average performance on real problems, which is the more crucial issue.

The two basic factors that determine the performance of an algorithm on a real problem are the *average computer time per iteration* and the *number of iterations*. Our next comparisons concern these factors.

Interior-point algorithms are far more complicated than the simplex method. Considerably more extensive computations are required for each iteration to find the next trial solution. Therefore, the computer time per iteration for an interior-point algorithm is many times longer than that for the simplex method. For fairly small problems, the numbers of iterations needed by an interior-point algorithm and by the simplex method tend to be somewhat comparable. For example, on a problem with 10 functional constraints, roughly 20 iterations would be typical for either kind of algorithm. Consequently, on problems of similar size, the total computer time for an interior-point algorithm will tend to be many times longer than that for the simplex method.

On the other hand, a key advantage of interior-point algorithms is that large problems do not require many more iterations than small problems. For example, a problem with 10,000 functional constraints probably will require well under 100 iterations. Even considering the very substantial computer time per iteration needed for a problem of this size, such a small number of iterations makes the problem quite tractable. By contrast, the simplex method might need 20,000 iterations and so might not finish within a reasonable amount of computer time. Therefore, interior-point algorithms often are faster than the simplex method for such huge problems.

FORMULATING VERY LARGE LINEAR PROGRAMMING MODELS

(Edited based on Introduction to Operations Research, 9th edition, Section 3.6)

Linear programming models come in many different sizes. For the examples in class, the model sizes range from three functional constraints and two decision variables up to dozens of functional constraints and decision variables. However, linear programming models in practice commonly have many hundreds or thousands of functional constraints. For example, the model for the United Airlines application often has over 20,000 decision variables. In fact, they occasionally will have even millions of functional constraints. The number of decision variables frequently is even larger than the number of functional constraints, and occasionally will range well into the millions.

Formulating such monstrously large models can be a daunting task. Even a "medium-sized" model with a thousand functional constraints and a thousand decision variables has over a million parameters (including the million coefficients in these constraints). It simply is not practical to write out the algebraic formulation, or even to fill in the parameters on a spreadsheet, for such a model.

So how are these very large models formulated in practice? It requires the use of a **modeling language**. How to solve these large-size models? It requires the use of a **solver**, which often applies the advanced methods, such as column generation, Blend's decomposition, and interior point methods.

Modeling Languages

A mathematical modeling language is software that has been specifically designed for efficiently formulating large mathematical models, including linear programming models. Even with millions of functional constraints, they typically are of a relatively few types. Similarly, the decision variables will fall into a small number of categories. Therefore, using large blocks of data in databases, a modeling language will use a single expression to simultaneously formulate all the constraints of the same type in terms of the variables of each type.

In addition to efficiently formulating large models, a modeling language will expedite a number of model management tasks, including accessing data, transforming data into model parameters, modifying the model whenever desired, and analyzing solutions from the model. It also may produce summary reports in the vernacular of the decision makers, as well as document the model's contents.

Several excellent modeling languages have been developed over the last couple of decades. These include AMPL, MPL, OPL, GAMS, and LINGO.

MPL (short for Mathematical Programming Language)is a product of Maximal Software, Inc. One feature is extensive support for Excel in MPL. This includes both importing and exporting Excel ranges from MPL. Full support also is provided for the Excel VBA macro language through OptiMax 2000. This product allows the user to fully integrate MPL models into Excel and solve with any of the powerful solvers that MPL supports, including **CPLEX**.ILOG, Inc.(developed CPLEX) also has introduced its own modeling language, called the *Optimization Programming Language (OPL),*that can be used with CPLEX to form the *OPL-CPLEX Development System*. (A trial version of that product is available at ILOG's website,www.ilog.com.)

CPLEX is an elite state-of-the-art software package that is widely used for solving large and challenging OR problems. When dealing with such problems, it is common to also use a modeling language to efficiently formulate the mathematical model and enter it into the computer.

MPL is a user-friendly modeling system that uses CPLEX as its main solver, but also has several other solvers, including LINDO, CoinMP (an open source solver that can solve larger linear programming and integer programming problems than the student version of CPLEX (which is limited to 300 constraints and variables)),

CONOPT, LGO, and BendX (useful for loving some stochastic models). A student version of MPL, along with the latest student version of CPLEX and its other solvers, is available free by downloading it from the Web, maximalsoftware.com. CPLEX 11 also extends beyond linear programming by including state-of-the-art algorithms for *integer programming* and *quadratic programming*, as well as *integer quadratic programming*.

Like MPL, **LINGO** is a powerful general-purpose modeling language and is a product of LINDO Systems, Inc.. A notable feature of LINGO is its great flexibility for dealing with a wide variety of OR problems in addition to linear programming. For example, when dealing with highly nonlinear models, it contains a global optimizer that will find a globally optimal solution. The latest LINGO also has a built-in compatible programming language so that you can do things like solve several different optimization problems as part of one run, which is particularly useful when doing parametric analysis.

LINDO (short for Linear, Interactive, and Discrete Optimizer) has an even longer history than CPLEX in the realm of applications of linear programming and its extensions. LINGO is a companion modeling language of LINDO, and also markets a spreadsheet-add-in optimizer called What's *Best!* that is designed for large industrial problems, as well as a callable subroutine library called the LINDO API. The LINGO software includes as a subset the LINDO interface that has been a popular introduction to linear programming for many people. The long-time popularity of LINDO is partially due to its ease of use. For "textbook-sized" problems, the model can be entered and solved in an intuitive, straightforward manner, so the LINDO interface provides a convenient tool for students to use. Although easy to use for small models, LINGO/LINDO can also solve large models, e.g., the largest version has solved real problems with 4 million variables and 2 million constraints. All of the LINDO Systems products, including the LINDO interface as a subset of the LINGO optimization modeling package, can be downloaded from LINDO Systems, www.lindo.com.

Available Software Options for Linear Programming

1. Excel and its Premium Solver for formulating and solving linear programming models in a spreadsheet format.
2. OPT/CPLEX or MPL/CPLEX for efficiently formulating and solving large linear programming models.
3. LINGO and its solver LINDO for an alternative way of efficiently formulating and solving large linear programming models.
4. Others such as Express-MP, a product of Dash Optimization (which now has joined Fair Isaac)
5. MATLAB and its optimizer.

Sensitivity Analysis

- **Graphical illustration**
- **Illustration based on the formulas in terms of matrices**
- **Reports from software**

Gathering accurate data frequently is difficult. Therefore, the value assigned to a parameter often is, of necessity, only a rough estimate. Because of the uncertainty about the true value of the parameter, it is important to analyze how the solution derived from the model would change (if at all) if the value assigned to the parameter were changed to other plausible values. This process is referred to as sensitivity analysis.

Sensitivity analysis is post-optimality analysis (analysis done after finding an optimal solution) is a very important part of most OR studies. This analysis is sometimes referred to as what-if analysis because it involves addressing some questions about what would happen to the optimal solution if different assumptions are made about future conditions.

Sensitivity analysis determines which parameters of the model are most critical, which in turn must be valued with special care to avoid distorting the output of the model. Since there is always some uncertainty in the data, it is useful to know over what range and under what conditions the components of a particular solution remain unchanged.

Further, the sensitivity of a solution to changes in the data gives us insight into possible technological improvements in the process being modeled. For instance, it might be that the available resources are not balanced properly and the primary issue is not to resolve the most effective allocation of these resources, but to investigate what additional resources should be acquired to eliminate possible bottlenecks. Sensitivity analysis provides an invaluable tool for addressing such issues.

Economic interpretation of the primal problem

In the models, **the constraints correspond to the resources, and the decision variables represent the activities.** The coefficients in the left-hand side of constraints (technological coefficients) represent the consumption/requirements of the resource by an activity (x_i). The b_i in right-hand side represents the available resource i. The resources associated with the nonbinding constraints are called free goods. The resources associated with the binding constraints are called scarce goods.

TABLE 6.6 Economic interpretation of the primal problem

Quantity	Interpretation
x_j	Level of activity j $(j = 1, 2, \ldots, n)$
c_j	Unit profit from activity j
Z	Total profit from all activities
b_i	Amount of resource i available $(i = 1, 2, \ldots, m)$
a_{ij}	Amount of resource i consumed by each unit of activity j

Shadow price (y*)

- The shadow price for the ith constraint of an LP is defined to be the amount by which the optimal z-value is improved if the rhs of the ith constraint is increased by 1. This definition applied only if the change in the rhs of the ith constraint leaves the current basis optimal.
- The **shadow price** for resource i (denoted by y_i*) measures the *marginal value* of this resource, i.e., the rate at which Z could be increased by (slightly) increasing the amount of this resource (b_i) being made available. **Individually increase any b_i by 1 indeed would increase the optimal value of Z by y_i*.**
- **The shadow price for nonbinding constraints is zero, implying increasing the resources associated with nonbinding constraints will not improve the optimal solution.**

Duality

- **Finding the dual of an LP**
- **Some basic duality theory (describing the relationships between LP and its dual)**
- **Shadow price and dual variables**

The following are the only possible **relationships** between the primal and dual problems.

1. If one problem has *feasible solutions* and a *bounded* objective function (and so has an optimal solution), then so does the other problem, so both the weak and strong duality properties are applicable.
2. If one problem has *feasible solutions* and an *unbounded* objective function (and so *no optimal solution*), then the other problem has *no feasible solutions.*
3. If one problem has *no feasible solutions,* then the other problem has either *no feasible solutions* or an *unbounded* objective function.

The *weak* and *strong duality properties* describe key relationships between the primal and dual problems. One useful application is for evaluating a proposed solution for the primal problem. For example, suppose that **x** is a feasible solution that has been proposed for implementation and that a feasible solution **y** has been found by inspection for the dual problem such that **cx =yb.** In this case, **x** must be *optimal* without the simplex method even being applied! Even if **cx <yb,** then **yb** still provides an upper bound on the optimal value of *Z,* so if **yb-cx** is small, intangible factors favoring **x** may lead to its selection without further ado.

One important application of duality theory is that the *dual* problem can be solved directly by the simplex method in order to identify an optimal solution for the primal problem. **The number of functional constraints affects the computational effort of the simplex method far more than the number of variables does**. If *m >n,* so that the dual problem has fewer functional constraints (*n*) than the primal problem (*m*), then applying the simplex method directly to the dual problem instead of the primal problem probably will achieve a substantial reduction in computational effort.

Dual simplex method

The dual simplex method requires that rof *begin and remain nonnegative* (optimality condition) while the right side *begins* with some *negative* values (subsequent iterations strive to reach a nonnegative right side (feasibility condition)). Consequently, this algorithm occasionally is used because it is more convenient to initialize in this form than in the form required by the simplex method. Furthermore, it frequently is used for reoptimization, because changes in the original model lead to some negative values of variables fitting this form.

Shadow prices and dual variables

- **The shadow price of the *i*th constraint of a *max* problem is the optimal value of the *i*th dual variable.** The shadow prices are the dual variables, so the shadow price for a ≤ constraint will be nonnegative; for a ≥ constraint, nonpositive; and for an equality constraint, unrestricted in sign.
- **The shadow price of the *i*th constraint of a *min* problem = – (the optimal value of the *i*th dual variable).**

Parametric Linear Programming

Sensitivity analysis involves changing one parameter at a time in the original model to check its effect on the optimal solution. By contrast, **parametric linear programming** (or **parametric programming** for short) involves the systematic study of how the optimal solution changes as *many* of the parameters change *simultaneously* over some range. This study can provide a very useful extension of sensitivity analysis, e.g., to check the effect of "correlated" parameters that change together due to exogenous factors such as the state of the economy. However, a more important application is the investigation of *trade-offs* in parameter values. For example, if the c_j values represent the unit profits of the respective activities, it may be possible to increase some of the c_j values at the expense of decreasing others by an appropriate shifting of personnel and equipment among activities. Similarly, if the b_i values represent the amounts of the respective resources being made available, it may be possible to increase some of the b_i values by agreeing to accept decreases in some of the others.

In some applications, the main purpose of the study is to determine the most appropriate trade-off between two basic factors, such as *costs* and *benefits*. The usual approach is to express one of these factors in the objective function (e.g., minimize total cost) and incorporate the other into the constraints (e.g., minimum acceptable level for the benefits). Parametric linear programming then enables systematic investigation of what happens when the initial tentative decision on the trade-off (e.g., the minimum acceptable level for the benefits) is changed by improving one factor at the expense of the other.

The algorithmic technique for parametric linear programming is a natural extension of that for sensitivity analysis, so it, too, is based on the simplex method.