MATH (160)

Mathematics for Data Analytics and Decision Making

Jesus De Loera

UC Davis, Mathematics

Monday, March 28, 2011

OPTIMIZATION MODELS

Let’s start with the simplest example possible

The Notip Table Company sells two models of its patented five-leg tables. The basic version uses a wood top, requires 0.6 hours to assemble, and sells for a profit of $200. The deluxe model takes 1.5 hours to assemble (because of its glass top), and sells for a profit of $350. Over the next week the company has 300 legs, 50 wood tops, 35 glass tops, and 63 hours of assembly available. Notip wishes to determine a maximum profit production plan assuming that everything produced can be sold.

Let's start with the simplest example possible

The Notip Table Company sells two models of its patented five-leg tables. The basic version uses a wood top, requires 0.6 hours to assemble, and sells for a profit of \$200. The deluxe model takes 1.5 hours to assemble (because of its glass top), and sells for a profit of \$350. Over the next week the company has 300 legs, 50 wood tops, 35 glass tops, and 63 hours of assembly available. Notip wishes to determine a maximum profit production plan assuming that everything produced can be sold.

We want a **mathematical model** for this problem.

We are using two variables to represent the decision on the production quantities:

x_1 – *number of basic tables,* x_2 – *number of deluxe tables*

$$\begin{array}{lll} \max & 200x_1 + 350x_2 & \text{(Total profit)} \\ \text{s.t.} & x_1 \leq 50 & \text{(Wood tops available)} \\ & x_2 \leq 35 & \text{(Glass tops available)} \\ & 5x_1 + 5x_2 \leq 300 & \text{(Legs available)} \\ & 0.6x_1 + 1.5x_2 \leq 63 & \text{(Hours of assembly time available)} \\ & x_1, x_2 \geq 0 & \text{(Non-negative quantities to be produced)} \\ & x_1, x_2 \in \mathbf{Z} & \text{(Integer quantities to be produced)} \end{array}$$

We study Mathematical Models Problems!

Mathematical Optimization Problem, abstract definition

$$\begin{array}{rl} \max & f(\mathbf{x}) \\ \text{s.t.} & \mathbf{x} \in F \subseteq X \end{array}$$

Here

- "max" means "maximize!"
- f is a function: the *objective function*
- "s.t." stands for "subject to (constraints)" or "such that"
- X is some set, the *space of solutions*
- $F \subseteq X$ is a subset: the set of all *feasible solutions*
- $\mathbf{x}$ is one feasible solution

F can be finite or infinite.

(Which optimization problems are easier to solve?)

Standard Form of Deterministic, Finite-Dimensional Mathematical Optimization Problems

$$\begin{aligned} \max(\min) \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_1(\mathbf{x}) \le 0 \\ & \vdots \\ & g_m(\mathbf{x}) \le 0 \\ & \mathbf{x} = (x_1, \ldots, x_n) \in X \end{aligned}$$

where $f, g_1, \ldots, g_m \colon \mathbf{R}^n \to \mathbf{R}$ are functions and $X = \mathbf{R}^n$, or $X = \mathbf{Z}^n$, or $X = \mathbf{R}^{n_1} \times \mathbf{Z}^{n_2}$.

We classify optimization problems according to the properties of the functions f and g_i and the space X.
Optimization Models allows us to make decisions and forecasting!

The problem with Brute Force

To solve a problem in practice on a computer, it is not enough to know that there exists an algorithm, i.e., a finite procedure.

It easily happens that the brute force method is not useful at all because for most interesting problems it takes too long, even on very fast computers.

Large bounds

If we have three integer variables x_1, x_2, x_3 with bounds $1 \leq x_i \leq 1\,000\,000$, we would have to check 10^{18} solutions!
If a computer with 8 processor cores running at $2.5\,\mathrm{GHz}$ could check one solution in one cycle, it would still take $10^{18}/(8 \cdot 2.5 \cdot 10^9) = 50 \cdot 10^6$ seconds (more than 1.5 years). This may be too long!

High dimension

Impressive examples appear when we have many variables, even if they have low bounds.In a combinatorial problem with 100 binary variables ($x_i = 0$ or $x_i = 1$), we would have to check 2^{100} solutions.A current supercomputer, IBM's Blue Gene/P, has 884,736 processors running at 850 MHz. It would take $1.6 \cdot 10^{15}$ years.The age of the universe is only $(13.7 \pm 0.1) \cdot 10^9$ years (Wikipedia).

How to solve this problem, II: Graphing

An important observation is that we can gain insight by graphing the example problem – because it only has 2 variables.

In R^2, we see that that each of the linear inequalities corresponds to a **half space**. Their intersection forms a **convex polygon** (a shape with finitely many corners, connected by straight edges), which we can easily draw.

The role of the objective function is slightly more complicated. For any $\alpha \in \mathbf{R}$, consider the **level set**

$$L_\alpha = \{\mathbf{x} = (x_1, x_2) \in \mathbf{R}^2 : 200x_1 + 350x_2 = \alpha\},$$

that is, the set of points of the plane where the objective function takes the value α. For each α, the level set is a line, and all the level sets are parallel to each other.

We now introduce vector notation. Let $\mathbf{a} = (200, 350)$. Then the equation of the level set reads $\langle \mathbf{a}, \mathbf{x} \rangle = \alpha$, where the left-hand side is the **inner product** (scalar product, dot product) of **a** and **x**. (You may know the notation $\mathbf{a} \cdot \mathbf{x}$, or $\vec{a} \cdot \vec{x}$, or $\mathbf{a}^\top \mathbf{x}$ (assuming **a** and **x** are written as column vectors).

Geometry of the problem

Now let $\mathbf{x}^0$ be any point in $\mathbf{R}^2$. Set $\alpha = 200x_1^0 + 350x_2^0$, then the point $\mathbf{x}^0$ lies on the level set L_α. Then we can write

$$\begin{aligned} L_\alpha &= \{\, \mathbf{x} = (x_1, x_2) \in \mathbf{R}^2 : \langle \mathbf{a}, \mathbf{x} \rangle = \langle \mathbf{a}, \mathbf{x}^0 \rangle \,\} \\ &= \{\, \mathbf{x} = (x_1, x_2) \in \mathbf{R}^2 : \langle \mathbf{a}, \mathbf{x} - \mathbf{x}^0 \rangle = 0 \,\}. \end{aligned}$$

Both equations will be called the **normal equation** of the line L_α.
Since a zero inner product indicates **orthogonality**, this means:

> *The level set consists of all points* $\mathbf{x}$ *such that the vector* $\mathbf{x} - \mathbf{x}^0$ *is orthogonal (**normal**) to the vector* $\mathbf{a}$*.*

Geometry of the problem, II

Now the optimization problem, if we **ignore the integrality constraints**, has the following **geometric interpretation**:

> *Among all the parallel lines L_α, find the one with the largest α such that the intersection of L_α with the feasible region (convex polygon) is nonempty.*

In our problem, we find that there is a unique optimal solution: For the optimal α, the intersection consists of a single point. (If we increase α even just a little bit beyond this, we push the line out of the feasible region.)
However, if we change the objective function a bit, it could happen that it is parallel to an edge, and then the intersection would then be the entire edge. Then the optimal solution is not uniquely determined; all solutions on that edge have the same objective function value and are thus optimal.

Geometry of the problem, III

This generalizes to **higher dimensions**:
In $\mathbf{R}^3$, the level set

$$\begin{aligned} L_\alpha &= \{\, \mathbf{x} = (x_1, x_2, x_3) \in \mathbf{R}^3 : \langle \mathbf{a}, \mathbf{x} \rangle = \langle \mathbf{a}, \mathbf{x}^0 \rangle \,\} \\ &= \{\, \mathbf{x} = (x_1, x_2) \in \mathbf{R}^2 : \langle \mathbf{a}, \mathbf{x} - \mathbf{x}^0 \rangle = 0 \,\}. \end{aligned}$$

is a plane; we thus see the normal equations of a plane in $\mathbf{R}^3$.
Then, for higher dimensions, we **define**:

Definition

A **hyperplane** in $\mathbf{R}^n$ is a set that can be described as

$$\{\, \mathbf{x} \in \mathbf{R}^n : \langle \mathbf{a}, \mathbf{x} - \mathbf{x}^0 \rangle = 0 \,\} \quad \text{for some } \mathbf{a} \in \mathbf{R}^n,\ \mathbf{a} \neq \mathbf{0},\ \mathbf{x}^0 \in \mathbf{R}^n,$$

or, equivalently, as

$$\{\, \mathbf{x} \in \mathbf{R}^n : \langle \mathbf{a}, \mathbf{x} \rangle = \alpha \,\} \quad \text{for some } \mathbf{a} \in \mathbf{R}^n,\ \mathbf{a} \neq \mathbf{0},\ \alpha \in \mathbf{R}.$$

Hyperplanes are $(n-1)$-dimensional objects in $\mathbf{R}^n$.

How to solve this problem, III: Using optimization software

We will be using three pieces of software in this lecture. (All of them are free at least for noncommercial and academic use. ZIMPL is free (open source) software.)

SCIP – “Solving Constraint and Integer Programs”

SCIP is a very good solver for **Mixed Integer Linear Optimization Problems**, i.e.,

$$\begin{aligned} \max \quad & f(\mathbf{x}) & \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 & \text{for } i = 1, \dots, m \\ & \mathbf{x} = (x_1, \dots, x_n) \in X & \end{aligned}$$

where f and g_i are linear functions, and $X = \mathbf{R}^n$, or $X = \mathbf{Z}^n$, or $X = \mathbf{R}^{n_1} \times \mathbf{Z}^{n_2}$.

SoPLEX – a solver for linear optimization problems

A super-fast solver for linear optimization problems ($X = \mathbf{R}^n$) is the basic building block of optimization and operations research technology.

ZIMPL – an algebraic modeling language for optimization

ZIMPL provides a convenient way to write down optimization problems in the computer.

SCIP uses both SoPLEX and ZIMPL internally.

The ZIMPL language: Variable definitions

A complete description is found in section 4.4 in ZIMPL User Guide.

Variable type:

`var NAME integer;`	Defines an integer variable
`var NAME binary;`	Defines a binary (0/1) variable
`var NAME real;`	Defines a real variable
`var NAME;`	Also defines a real variable

All variables (except binary) have a lower bound of 0 and no upper bound. If different upper bounds are needed, change the variable definition:

`var NAME integer >= -5 <= 5`	Defines an integer variable that can take values from -5 to 5
`var NAME real >= -infinity <= infinity`	Defines a free variable that can take arbitrary real values

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

Each of the constraints is written as:

```
subto NAME: TERM SENSE TERM
```

Here SENSE is one of $<=$, $>=$, $==$.

(The ZIMPL language has a much greater power than this; we'll dive into its complexity gradually.)

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

Each of the constraints is written as:

```
subto NAME: TERM SENSE TERM
```

Here SENSE is one of $<=$, $>=$, $==$.

(The ZIMPL language has a much greater power than this; we'll dive into its complexity gradually.)

Our example problem in ZIMPL

ZIMPL input files are plain text files; use a text editor (such as *Emacs* or *gedit*) to create them, not a word processor.

```
var basictables integer;
var deluxetables integer;

maximize profit:
200*basictables + 350*deluxetables;

subto legsconstraint:
5*(basictables + deluxetables) <= 300;

subto woodtopsconstraint:
basictables <= 50;

subto glasstopsconstraint:
deluxetables <= 35;

subto assemblyhoursconstraint:
0.6*basictables + 1.5*deluxetables <= 63;
```

The ZIMPL language: Variable definitions

A complete description is found in section 4.4 in ZIMPL User Guide.

Variable type:

`var NAME integer;`	Defines an integer variable
`var NAME binary;`	Defines a binary (0/1) variable
`var NAME real;`	Defines a real variable
`var NAME;`	Also defines a real variable

The ZIMPL language: Variable definitions

A complete description is found in section 4.4 in ZIMPL User Guide.

Variable type:

`var NAME integer;`	Defines an integer variable
`var NAME binary;`	Defines a binary (0/1) variable
`var NAME real;`	Defines a real variable
`var NAME;`	Also defines a real variable

All variables (except binary) have a lower bound of 0 and no upper bound. If different upper bounds are needed, change the variable definition:

`var NAME integer >= -5 <= 5`	Defines an integer variable that can take values from -5 to 5
`var NAME real >= -infinity <= infinity`	Defines a free variable that can take arbitrary real values

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

Each of the constraints is written as:

```
subto NAME: TERM SENSE TERM
```

Here SENSE is one of $<=$, $>=$, $==$.

The ZIMPL language: Objective and constraints

The objective function is written as:

```
maximize NAME: TERM;
minimize NAME: TERM;
```

where TERM is an arbitrary (linear) expression.

Each of the constraints is written as:

```
subto NAME: TERM SENSE TERM
```

Here SENSE is one of $<=$, $>=$, $==$.

(The ZIMPL language has a much greater power than this; we'll dive into its complexity gradually.)

Some combinatorial optimization: The stable set problem

Given a graph $G = (V, E)$, find a maximum-cardinality vertex set so that no two selected vertices are connected by an edge.

Variables?

Constraints (inequalities)?

Objective function?

The Traveling Salesperson Problem

Traveling Salesperson Problem (TSP)

A traveling salesperson needs to visit *n* cities, starting from city 1, visiting each other city exactly once, and returning to city 1. Find the shortest possible such tour.

"Distances" between the cities can be measured in different ways:

- If cities are considered as points $(x_i, y_i) \in \mathbf{R}^2$, a useful distance between cities could be the Euclidean distance
$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$
- Other distances that make sense are: Number of miles on the shortest road connection between cities *i* and *j*, airfares in US dollars, ...
Some of these distance functions *d* have some of these properties:
  - Satisfies triangle inequality (Metric TSP):
$$d(i,j) \leq d(i,k) + d(k,j)$$
  - Symmetry (Symmetric vs. Asymmetric TSP):
$$d(i,j) = d(j,i)$$

Some comments on the TSP

The TSP is one of the most famous combinatorial optimization problems.

- It appears in a wide array of applications.
    - Various route planning problems
    - Printed circuit board drilling
    - Application in genome sequencing

Some comments on the TSP

The TSP is one of the most famous combinatorial optimization problems.

- It appears in a wide array of applications.
  - Various route planning problems
  - Printed circuit board drilling
  - Application in genome sequencing
- It is notoriously hard to solve.
  - A symmetric TSP has $(n-1)!/2$ possible tours, so brute force is not feasible.
  - Simple, intuitive heuristics such as the “visit-nearest-neighbor algorithm” fail spectacularly
  - No “efficient” algorithm is known for it today, and it belongs to a huge class (*NP-hard*) of other, seemingly unrelated, difficult problems that we could suddenly solve efficiently, once an efficient algorithm for TSP became available.
- We’ll find out how well our black-box optimization solver works for this model

Excellent web resource, by Prof. William Cook (Georgia Tech):

`http://www.tsp.gatech.edu/`

- Concorde (state-of-the-art software for TSP), world records on the TSP, applications, and illustrations.

Modeling the TSP as a standard optimization problem

Variables?

Constraints (inequalities)?

Modeling the TSP as a standard optimization problem, I

- A key observation is that every tour of the TSP on n cities can be viewed as a subgraph (V, E') of the complete graph $K_n = (V, E)$ on n nodes.
  (We disregard orientation and starting point of the tour by doing so.)
- Remember that the edges of the complete graph are the 2-element subsets of n:
  $$E = \{\, \{i,j\} : i,j = 1, \dots, n \,\}$$
  Note that the order of i and j does not play any role:
  $$\{i,j\} = \{j,i\}$$
- We can "encode" any subgraph (V, E') with a "set" of 0/1 variables, one for each edge of the complete graph:
  $$x_{\{i,j\}} = \begin{cases} 1 & \text{if edge } \{i,j\} \text{ is present, i.e., } \{i,j\} \in E' \\ 0 & \text{if edge } \{i,j\} \text{ is not present} \end{cases}$$
- Thus, each vector $(x_{\{i,j\}})_{\{i,j\}\in E} \in \{0,1\}^E$ "encodes" a subgraph (V, E'). One way of writing this vector is as the upper triangle of a square $n \times n$ matrix – but how we write it, is not essential.
- Important is that this is a one-to-one correspondence between the combinatorial objects ("subgraphs") and 0-1-vectors.

Modeling the TSP as a standard optimization problem, II

Next we wish to express the objective function.

- We wish to minimize the total length of the tour T, which we view as the edge set of a subgraph (V, T) of the complete graph $K_n = (V, E)$.
- Using the notation $d(i, j)$ for the length of way from city i to j (or reversely – remember we deal with the symmetric case!), the total length is:

  $$length(T) = \sum_{\{i,j\} \in T} d(i, j)$$

  This summation is not "nice" – its domain of summation depends on the solution T. We prefer to sum over fixed domains of summation!
- Now we remember that we have 0/1 variables $x_{\{i,j\}}$ that are 1 if $\{i, j\} \in T$, and 0 otherwise. So it does not change anything if we multiply $d(i, j)$ by $x_{\{i,j\}}$:

  $$length(T) = \sum_{\{i,j\} \in T} x_{\{i,j\}} d(i, j) = \sum_{\{i,j\} \in E} x_{\{i,j\}} d(i, j)$$

  In the last step, we extended the domain of summation to all edges in E – again, nothing happens, since all the added summands are 0.
- So now we have expressed the length of a tour as a linear function in our $x_{\{i,j\}}$ variables; note the domain of summation is independent of the tour!

Modeling the TSP as a standard optimization problem, III

- **Since all tours are subgraphs, but not all subgraphs are tours, we need to add constraints on our variables, to make sure that only tours are feasible solutions.**
- Remember that we call a vertex v and an edge e **incident** if $v \in e$, i.e., v is one of the endpoints of the edge e.
- The **degree** of a vertex v is the number of edges incident with it.
- A key observation is that in a tour, viewed as a subgraph of K_n, every vertex has degree 2 (if we oriented the tour, one edge would go in, one edge would go out).
- So let's write down this insight as a constraint, for every vertex i:

$$\sum_{j:\{i,j\}\in E} x_{\{i,j\}} = 2.$$

Again, the domain of summation is independent of the tour, so this equation is a linear constraint in our variables $x_{\{i,j\}}$.

Modeling the TSP as a standard optimization problem, III

- **Since all tours are subgraphs, but not all subgraphs are tours, we need to add constraints on our variables, to make sure that only tours are feasible solutions.**
- Remember that we call a vertex v and an edge e **incident** if $v \in e$, i.e., v is one of the endpoints of the edge e.
- The **degree** of a vertex v is the number of edges incident with it.
- A key observation is that in a tour, viewed as a subgraph of K_n, every vertex has degree 2 (if we oriented the tour, one edge would go in, one edge would go out).
- So let's write down this insight as a constraint, for every vertex i:

$$\sum_{j:\{i,j\}\in E} x_{\{i,j\}} = 2.$$

Again, the domain of summation is independent of the tour, so this equation is a linear constraint in our variables $x_{\{i,j\}}$.

Putting the TSP model in the computer

- In the computer, it is convenient to represent edges (2-element sets) $\{i,j\}$ as (ordered) pairs (i,j) with $i < j$.
- Thus, for a 6-city TSP, we would be using variables named `x12`, `x13`, `x14`, `x15`, `x16`, `x23`, `x24`, `x25`, `x26`, `x34`, `x35`, `x36`, `x45`, `x46`, `x56`
- When we write down the constraint

$$\sum_{j:\{i,j\}\in E} x_{\{i,j\}} = 2,$$

by using the ordered-pair representation, we actually write

$$\sum_{j<i} x_{(j,i)} + \sum_{j>i} x_{(i,j)} = 2.$$

- The resulting optimization model in ZIMPL is found in the file `tsp6-1.zpl`

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

  $$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

  This does not look like a tour! What are we missing?

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

  $$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

  This does not look like a tour! What are we missing?
- We are not missing anything; to the contrary, we have **too much**!
  Our integer program has many feasible solutions that do not correspond to tours.
  (The corresponding subgraphs do satisfy the degree-2 conditions.)
  In the example, we obtained a feasible solution that corresponds to 2 cycles of length 3.

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

  $$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

  This does not look like a tour! What are we missing?
- We are not missing anything; to the contrary, we have **too much**!
  Our integer program has many feasible solutions that do not correspond to tours.
  (The corresponding subgraphs do satisfy the degree-2 conditions.)
  In the example, we obtained a feasible solution that corresponds to 2 cycles of length 3.
- Dually speaking, we **are** missing something: We need to add more inequalities that “forbid” short cycles.

TSP Formulation – What are we missing?

- Using SCIP on `tsp6-1.zpl` or `tsp6-2.zpl`, we obtain an optimal solution of

  $$x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1, \quad \text{all other } x_{ij} = 0$$

  This does not look like a tour! What are we missing?
- We are not missing anything; to the contrary, we have **too much**!
  Our integer program has many feasible solutions that do not correspond to tours.
  (The corresponding subgraphs do satisfy the degree-2 conditions.)
  In the example, we obtained a feasible solution that corresponds to 2 cycles of length 3.
- Dually speaking, we **are** missing something: We need to add more inequalities that "forbid" short cycles.

Lemma

Let $T \subseteq E$ be any TSP tour on K_n.
Let $S \subseteq V$ be a vertex subset of size $3 \leq |S| \leq n-3$. Then

$$|\{\,\{i,j\} \in T : i,j \in S\,\}| \leq |S| - 1.$$

Complete TSP Formulation

Theorem (Complete TSP Formulation)

The 0/1 solutions of the system

$$\sum_{j:\{i,j\}\in E} x_{\{i,j\}} = 2 \qquad \textit{for all vertices } i = 1,\ldots,n$$

$$\sum_{\substack{\{i,j\}\in E:\\ i,j\in S}} x_{\{i,j\}} \le |S| - 1 \qquad \textit{for all } S \textit{ in } K_n \textit{ with } 3 \le |S| \le n-3$$

are in one-to-one-correspondence with the TSP tours on K_n.

- How many short-cycle inequalities?

$$2^n - 2\binom{n}{0} - 2\binom{n}{1} - 2\binom{n}{2}$$

For $n = 15$: about 32000.

But how do we solve the TSP model?

This formulation is the basis for the state-of-the-art approaches (“branch and cut”) to the TSP, which:

- Repeatedly solve a linear relaxation (where the integrality of the variables is ignored) – a linear optimization problem
- Branch on subproblems where some variables are fixed to 0 or 1
- Use linear relaxation bounds to throw away subproblems where no good solution exists
- Add violated inequalities (short-cycle inequalities and “stronger” inequalities)

It motivates us in this class to:

- Learn expressive modeling languages (ZIMPL) that allow us to add a large number of well-structured inequalities without writing each of them by hand
- Study fast algorithms for solving discrete optimization problems, even if the number of variables and constraints is very large.

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Modeling using Mathematical Programs

Solving Integer Programs

Solving Integer Programs

Solving Integer Programs