



Learning Mathematical Programming for IBM ILOG OPL V6.3

Student workbook

© Copyright IBM Corporation 2009. All rights reserved.

IBM, the IBM logo, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others. References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

Contents

Lesson 1: The Big Picture	9
What is operations research?.....	10
What is optimization?.....	13
OPL.....	20
Summary.....	21
Lesson 2: Linear Programming	23
Introduction to linear programming.....	24
Example: a production problem.....	26
Algorithms for solving LP problems.....	36
Summary.....	43
Lesson 3: Beyond Simple LP	45
Network models.....	46
Nonlinearity and convexity.....	53
Piecewise Linear Programming.....	55
Integer optimization.....	57
Quadratic programming.....	67
Summary.....	69
Lesson 4: Modeling Practice	71
Modeling in the real world.....	72
The importance of sparsity.....	74
A packing model for “what-if” analysis.....	76
Tips for better models.....	79
Summary.....	80
Conclusion	3

About this course

This section provides you with a brief description of the course, audience, suggested prerequisites, and course objectives.

Course description

This is a one-day course on Mathematical Programming (MP) that prepares students to work with IBM® ILOG OPL. This course provides the fundamentals necessary to get started with OPL and to construct simple optimization models.

Audience

Professionals who want to learn how to use OPL to solve business optimization problems, for example developers, IT specialists and business managers who may be working in collaboration with Operations Research (OR) experts.

Course objectives

- Explain what Operations Research (OR) is.
- Explain what mathematical optimization is.
- Explain what mathematical programming is.
- Explain the differences between LP, IP, MIP, NLP, and QP.
- List some of the algorithms used to solve LPs.
- Describe what a network model is and the benefits of using network models.
- Describe what is meant by convexity and nonlinearity.
- Describe what a piece-wise linear model is and when this is useful.
- Explain under which circumstances integer variables are required, and why this could increase computational effort.
- Explain what is meant by sparsity and how this is relevant in models and data.
- Explain how uncertainty, periodicity, and preferences can be dealt with in optimization models.
- Use OPL to build a simple optimization model and perform “what-if” analysis.
- Be comfortable enough with MP and OPL to continue on to the three-day *Optimization Modeling with IBM ILOG OPL* training course.
- Be able to converse with an OR expert on the topics covered in this course.

Prerequisites

- Working knowledge of the Microsoft Windows operating system
- Knowledge of basic algebra

How to use




Training materials

- Printed student workbook
- Slide presentation used by the instructor
- Workshop (already installed on your computer)
- OPL labs, with solutions, to follow by using the steps in the workshop (already installed on your computer)

This workbook covers the basic concepts of Mathematical Programming necessary to achieve the course objectives. The instructor will go through the material in sequence as it is presented in the workbook, adding examples and explanations as appropriate.

The workbook includes all the slide content, as well as some additional explanatory text after some slide content. In addition, the workbook includes some white space for taking notes.

Some lessons contain practices. Practices can be any of the following:

-  Labs (indicated by a small computer icon) are exercises that have associated workshop instructions and OPL files. Workshop instructions are accessible from your web browser and repeated in this workbook (in italics), so you can follow the instructions either on your screen or in the book. You will perform these exercises with some instructor guidance.
-  Hands-on items (indicated with a small person-at-computer icon) are less formal demonstrations and exercises that don't require workshop instructions. They can be carried out by both the instructor and the students, or just used by the instructor to demonstrate a particular feature.
-  Discussions (indicated with a small icon of two people talking) are informal and are used to further explore a topic through discussion with fellow-students and the instructor.

<TrainingDir> refers to the directory in which training exercises are stored. Usually this is a directory that can be found directly on C:, for example: **C:\OPLTraining**.

Using the documentation

The OPL user documentation is available for reference at any point during the training. Although this course provides some basic OPL syntax explanations, you may find the documentation helpful to better understand the examples and exercise problems. The documentation can be found within the standard OPL Online Help which can be accessed from the OPL IDE by clicking **Help > Help Contents** on the toolbar. Contextual help for keywords in OPL (keywords are shown in color in the IDE) is shown in a Help window inside the OPL IDE when highlighting the keyword, either by dragging the mouse over it or double-clicking it. Note that you first have to open the OPL Online Help for this to work.

Using the .html workshop

The instructor will show you how to use the .html workshop to access the lab instructions. These instructions are repeated in the workbook, but the .html workshop contains useful links to the documentation.

Lesson 1: The Big Picture

At the end of this lesson you will be able to:

- Explain the terms operations research, optimization and mathematical programming.
- List some ideas of how optimization can benefit you and your organization.
- Describe a typical optimization model development cycle.
- Describe the basic elements of an optimization model.
- Understand what OPL is, and what types of optimization techniques it offers.

Why learn about Mathematical Programming (MP)?

Mathematical optimization is a powerful Operations Research (OR) decision making technique, often used as part of business decision applications.

There are several reasons to learn about mathematical optimization, for example:

- You are part of a team developing an optimization-based application
- You are a business manager who needs to converse with an optimization application development team
- You are a user of an optimization-based application and would like to better understand how optimization works

IBM[®] ILOG OPL includes the following mathematical optimization techniques:

- Mathematical Programming (MP)
- Constraint Programming (CP)

This course introduces MP and OPL.

This course covers OPL on a very basic level. The more advanced 3-day OPL course teaches how to effectively design and construct models with OPL.

What is operations research?

Learning objective

Gain insight into the background and history of the OR profession and where it stands today.

Key term

Operations Research (OR)

Operations Research in a nutshell

“Operations Research (OR) is all about doing more with the data you have and making better decisions. Businesses are driven by data, but don't always know how to use it to competitive advantage. OR is a mathematical approach, and its software tools can help CIOs know what to do with data once it's been analyzed... With global competition intensifying, businesses need to coordinate their operations better. OR can offer better real-time decision-making and data to do analysis and forecasting. IT can

show better ROI and more measurable results – meaning better value and service to the business.”

-Irv Lustig, director, IBM ILOG direct channel

The origins of OR

The field of OR was first developed during World War II, when scientists helped the military to solve complex logistical and strategical problems related to military resource deployment and operations planning.

The name, *Operations Research*, or, sometimes, *Operational Research*, stems from its application to military *operations*.

After the war, scientist started applying Operations Research techniques to similar problems in industry.

In the 1950s, some OR experts began using the term “Management Science” to describe what they did, preferring to avoid the military connotations of the term “Operations Research.” Today, the two terms are essentially considered to be synonymous.

Significant researchers in the United Kingdom include Patrick Blackett, Cecil Gordon, C. H. Waddington, Owen Wansbrough-Jones and Frank Yates. In the United States, important pioneers include Phillip M. Morse, George Kimball,, Russell Ackoff, C. West Churchman, Merrill Flood , Melvin Salveson, and Andrew Vazsonyi, among many others. One name in particular, George B. Dantzig, stands out in the history of operations research.

OR tools

OR professionals use a variety of tools, including:

- Mathematical optimization
- Statistics
- Probability theory
- Simulation
- Decision analysis

These tools are often used in combination, but this training focuses on mathematical optimization.

Linear Programming (LP) is probably the best-known mathematical optimization technique, and many other optimization methods have been derived from LP.

George B. Dantzig and the simplex method

Dr. George B. Dantzig is generally considered to be the "father" of linear programming.

While working at the Pentagon, in the 1950s, as mathematical advisor to the comptroller of the U.S. Air Force, he formulated the general linear programming problem and developed the simplex algorithm for solving it.

Dr. Dantzig applied these techniques to Air Force deployment and logistical planning problems.

After the war he continued his pioneering work as a researcher and academic.

During his extensive academic career at both Berkeley and Stanford, Professor Dantzig demonstrated how to exploit linear programming's generality, notably in these areas:

- Formulation and solution of management problems in most major industries
- Resolution of strategic and tactical problems in defense
- Evaluation of plans and operational solutions of national and world-wide energy and other resource limited problems
- Theoretical advances in mathematics, statistics, economics, game theory and computer science
- Adaptation of the linear programming model to a more general set of mathematical programming problems

The simplex method is the basis for solving a large number of optimization problems, and a revised simplex algorithm is central to the CPLEX solver engine used by OPL.

OR today

Most OR professionals today perform activities in one or more of the following areas:

- Helping organization improve their business processes and performance
- Developing and deploying models and algorithms for information systems and applications
- Contributing to the debate about the purpose and direction of institutions and policies

Companies continue to have dedicated OR departments, but more and more, people with OR skills and training are found inside diverse departments such as finance, planning, transportation, and production. These people are sometimes called OR analysts.

Another current trend is the development of specialist external consulting organizations who are called upon by companies to perform a specific OR-related analysis.

The relationship between OR and IT

OR applications often require:

- large amounts of historic and real-time data
- tight integration with other business systems
- powerful mathematical processing capabilities

Computers and Information Technology (IT) help address these needs with, for example:

- database design and management
- computer network design
- system administration

IT developers and OR experts collaborate frequently in designing and developing sophisticated optimization-based implementations.

A related product, IBM[®] ILOG ODM Enterprise, offers a collaborative development and user environment for custom optimization-based decision applications. ODM

Enterprise facilitates collaboration between OR experts, IT developers, and business users, and includes OPL for optimization model development.

References

Part of this content comes from the INFORMS 2009 website, specifically "George Dantzig-A Biography", downloaded from http://www2.informs.org/History/dantzig/bio_achieve.htm on Aug 31, 2009.

INFORMS = Institute for Operations Research and the Management Sciences.

What is optimization?

Learning objective

Gain a basic understanding of optimization and its applications.

Key terms

- mathematical programming
- constraint programming
- objective
- decision variable
- constraint
- optimization engine
- feasible solution
- optimal solution

How does optimization work?

In the current context, the term “optimization” refers to mathematical optimization. This implies using a set of mathematical techniques to find the best possible solution to a business problem. An optimization-based solution involves:

- An optimization model, defined in terms of an objective, decision variables, and constraints
- Data to create an instance of the model
- An optimization engine (or algorithm) to solve the model instance

The purpose of the optimization process is to find values for the decision variables that satisfy all constraints and optimize the objective function.

Optimization-based applications

Optimization-based decision applications involve embedded optimization models and algorithms as part of larger applications that might include, for example, a graphical user interface and integration with databases and other enterprise systems.

Such applications can be used to perform “what-if” analysis: users can compare different scenarios, each involving different data.

By invoking the optimization model interactively and repeatedly, users can judge the impact of data changes, and analyze trade-offs between conflicting business goals and constraints.

What can be optimized?

Optimization is a decision support tool, that has applications in virtually every industry, as shown in the following diagram.

Common Optimization Applications

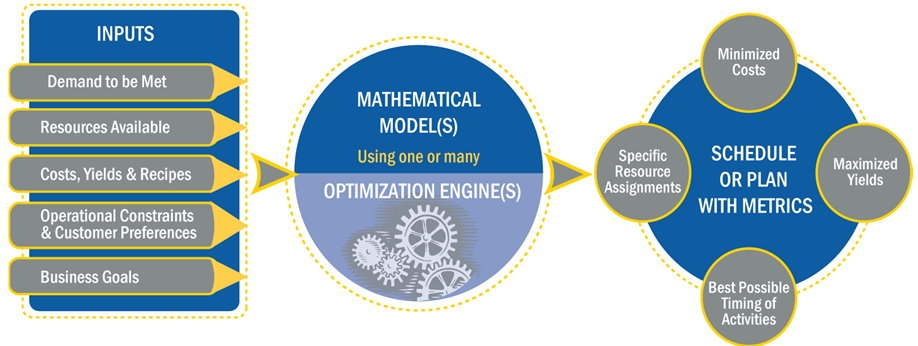
MANUFACTURING	TRANSPORTATION & LOGISTICS	FINANCIAL SERVICES	UTILITIES, ENERGY & NATURAL RESOURCES	TELECOM	MULTIPLE/ OTHER
<ul style="list-style-type: none"> • Inventory optimization • Supply chain network design • Production planning • Detailed scheduling • Shipment planning • Truck loading • Maintenance scheduling 	<ul style="list-style-type: none"> • Depot/warehouse location • Fleet assignment • Network design • Vehicle & container loading • Vehicle routing & delivery scheduling • Yard, crew, driver & maintenance scheduling • Inventory optimization 	<ul style="list-style-type: none"> • Portfolio optimization and rebalancing • Portfolio in-kinding • Trade crossing • Loan pooling • Product/price recommendations 	<ul style="list-style-type: none"> • Supply portfolio planning • Power generation scheduling • Distribution planning • Water reservoir management • Mine operations • Timber harvesting 	<ul style="list-style-type: none"> • Network capacity planning • Routing • Adaptive network configuration • Antenna and concentrator location • Equipment and service configuration 	<ul style="list-style-type: none"> • Workforce scheduling • Advertising scheduling • Marketing campaign optimization • Revenue/Yield management • Appointment & field service scheduling • Combinatorial auctions for procurement

A wide range of decisions can be optimized - for example:

- How many products to make, when and how
- How to transport goods, people, materials
- How to blend raw materials
- How to schedule people, tasks, machines
- How to locate and allocate facilities, equipment
- How to spend capital

The following diagram illustrates some of the types of information you can consider, and what results are produced by the optimization process.

Optimization Inputs and Outputs



Practice



Discuss some ideas among yourselves and with the instructor on what can be optimized in your organization, and what benefits you would expect from the optimization process.

A typical optimization model development cycle

The following tasks are part of this cycle:

1. Define the scope of the problem in business terms.
2. Identify the objective(s), decision variables and constraints.
3. Construct the mathematical optimization model.
4. Gather data to populate the model.
5. Create and test a prototype with a small data set.
6. Edit the model if required, based on the test results.
7. Create and test a complete instance of the model.
8. Evaluate performance and adjust the model as needed.
9. Develop advanced solution techniques, if required.

Optimization techniques

Two techniques for mathematical optimization that can be applied by using the IBM® ILOG optimization products are:

- Mathematical Programming (MP)
- Constraint Programming (CP)

Mathematical programming

Mathematical programming techniques include:

- Linear Programming (LP)
- Integer Programming (IP)
- Mixed-Integer Programming (MIP)
- Quadratic Programming (QP)
- Mixed-Integer Quadratic Programming (MIQP)
- Several others, some of which are not currently offered in the IBM® ILOG product line, for example Non-Linear Programming (NLP) in a broader sense (strictly speaking QP, which is offered by IBM ILOG, is a subset of NLP).

The different techniques address different types of problem structures, with LP being the original and generally simplest of these.

Constraint programming

Constraint programming is especially appropriate for:

- detailed scheduling problems
- certain combinatorial optimization problems that cannot be easily linearized and solved using traditional MP methods

The objective function

The objective function of an optimization model is a mathematical representation of the business objectives or goals to be achieved. Objective functions always start with the words “maximize” or “minimize”, and can be either a simple expression involving a single business objective, or a more complex expression combining several business objectives.

Some examples of objectives are:

- Maximize profit

- Minimize cost
- Minimize tardiness
- Maximize customer service

What are some of the objectives you'd like to optimize in your business?

Decision variables

Decision variables in an optimization model represent values or decisions that can be changed by the solver engine in order to arrive at the best possible value of the objective function.

Examples of decision variables are:

- How much of a given product to produce
- How many people to hire for a given task
- Which location to choose for a new warehouse
- The start time of a given task

Can you think of some decision variables that could influence the objectives you thought of?

Some characteristics of decision variables are:

- Their value is determined during the solution process
- They are initially unknown in a model, although it's possible to provide an initial guess to help the optimization
- Decision variables have a domain: the set of all possible values for the variable
- Decision variables have bounds: the lower and upper limits of the domain (infinite bounds are allowed)
- Decision variables have a type, for example real, integer, Boolean, or interval (used in CP Optimizer only)

It is important to choose the decision variables well, because they impact the formulation of the constraints and the solution method used. For example, LPs may only use real variables, whereas IPs or MIPs allow integers.



The word “variable” alone, when used in an MP or CP context, refers to a decision variable. Note that this is different from a **script variable** (used in IBM ILOG Script) or a **data element** that represents data in an OPL model.

Constraints

Constraints define the relationships between different decision variables and also relate the data to the decision variables. They represent the limits within which the solution should exist.

Examples of constraints are:

- The amount produced of a given product should be less than the production capacity
- The people hired for a given task should have a minimum set of skills
- The total cost of building a new warehouse should be less than the budgeted amount
- The output from a process should equal the yield multiplied by the input
- A certain task can only be started once a related task has been completed

What are some of the constraints that describe your business problem?

Hard versus soft constraints

Constraints are either “hard” or “soft”.

Hard constraints should not be violated under any circumstances, for example a constraint on production capacity when no additional capacity is available by any means.

Soft constraints can be violated under certain circumstances, for example a constraint that requires production to exceed customer demand. Such a constraint may need to be violated if there is, for example, an unexpected surge in demand or an equipment failure.

In the case of soft constraints, a penalty is often added to the objective to minimize the constraint violation.

Try and identify whether the constraints you mentioned for your business problem are hard or soft.

Solving optimization problems

The following solver engines are available through IBM ILOG:

- IBM® ILOG CPLEX – solves MP problems, specifically LPs, MIPs and some QP problems
- IBM ILOG CP Optimizer – solves CP problems (IBM ILOG's second generation CP engine)
- IBM ILOG CP – IBM ILOG's first generation CP engine used to solve complex CP problems beyond the current capabilities of CP Optimizer, for example complex routing problems

Advanced solver engines such as CPLEX and CP Optimizer can automatically choose the best solution algorithm for a particular problem, based on the problem structure.

It's possible to have more control over the algorithm by setting some parameters, or by developing custom algorithms that use some of these solvers' capabilities.

IBM ILOG CP is not accessible through OPL.

Feasibility vs. optimality

In optimization, a solution is a proposal of specific values for the decision variables, regardless of whether it is a desirable or even an allowable set of choices. Solutions can be classified as:

- **Feasible:** A solution that satisfies all the constraints
- **Optimal:** A feasible solution that also achieves the best possible value of the objective function
- **Infeasible:** A solution where one or more constraints are violated
- **Unbounded:** A solution to a problem where the constraints are not completely described, resulting in an objective that goes to plus or minus infinity, depending on whether it's a maximization or minimization problem.

The set of all feasible solutions is called the **feasible region**.

Note that even when an optimization engine finds an optimal solution, the human factor is often still very important in evaluating the solution.

This page is intentionally left blank.

OPL

Learning objective

Acquire an overview of OPL

Key term

OPL

What is OPL?

ILOG OPL Development Studio...

...is an **IDE** (Integrated Development Environment)

that uses **OPL** (Optimization Programming Language)

to build and execute mathematical **models**

composed of **objective functions** and **constraints**

that read **data**

to be **optimized**

OPL has four components:

- Optimization Programming Language – a declarative high-level optimization language
- The OPL IDE
- IBM[®] ILOG Script (a scripting language for OPL)
- APIs that allow you to embed OPL models into standalone applications, written in C++, .net or Java[™].

The following IBM[®] ILOG optimization engines are accessible through OPL:

- IBM ILOG CPLEX for Mathematical Programming (MP)
- IBM ILOG CP Optimizer for Constraint Programming (CP)

OPL also provides a direct connection to IBM ILOG Optimization Decision Manager (ODM) that allows the creation of runtime applications for business users.

Summary

Review

Business decisions often require the involvement of many different people in different roles, as well as analysis of complex data and interactions. Operations research is a discipline focused on making better decisions by using a variety of mathematical techniques, and mathematical optimization is one of the techniques used by operations research professionals. Mathematical programming is one of the techniques used to perform mathematical optimization, and includes, for example, linear programming, integer programming, mixed-integer programming and quadratic programming.

A typical optimization model development cycle involves the following tasks:

1. Define the scope of the problem in business terms.
2. Identify the objective(s), decision variables and constraints.
3. Construct the mathematical optimization model.
4. Gather data to populate the model.
5. Create and test a prototype with a small data set.
6. Edit the model if required, based on the test results.
7. Create and test a complete instance of the model.
8. Evaluate performance and adjust the model and/or develop advanced solution techniques, if required.

The basic elements of an optimization model are the data, decision variables, objective function, and constraints. Data is input into the model and is usually not changed during the solution process. The objective function is a mathematical representation of the business objectives or goals to be achieved. Decision variables represent values or decisions that can be changed by the solver engine in order to arrive at the best possible value of the objective function. Constraints define the relationships between different decision variables and also relate the data to the decision variables. They represent the limits within which the solution should exist.

OPL is an integrated development environment that uses a declarative optimization programming language to build and execute mathematical optimization models. OPL uses the CPLEX solution engine for MP problems and the CP Optimizer engine for CP problems.

This page is intentionally left blank.

Lesson 2: Linear Programming

At the end of this lesson you will be able to:

- Describe the characteristics of an LP in terms of the objective, decision variables and constraints.
- Formulate a simple LP model on paper.
- Conceptually explain the following terms in the context of LP:
 - dual
 - feasible region
 - infeasible
 - unbounded
 - slack
 - reduced cost
 - degenerate
- Describe some of the algorithms used to solve LPs.
- Explain what presolve does.

Introduction to linear programming

Learning objective

Understand the basic characteristics of a linear program.

Key terms

- linear programming
- linear expression
- linear constraint
- objective function
- continuous variable

What is linear programming?

Linear programming (LP) deals with the maximization (or minimization) of a linear **objective function**, subject to **linear constraints** where all the decision variables are **continuous**. The linear objective and constraints consist of **linear expressions**.

A **linear expression** is a scalar product, for example, the expression

$$a(0) + a(1)x(1) + \dots + a(n)x(n).$$

In this expression, by convention:

- $a(i)$ represents constants (data instances)
- $x(i)$ represents variables (unknowns)

This expression can also be written as $\mathbf{a}^T \mathbf{x}$, where \mathbf{a} is the vector of constants and \mathbf{x} is the vector of variables.

Nonlinear terms that involve variables (such as x and y) are not allowed in linear expressions, for example the following are not allowed:

- Multiplication of two or more variables ($\mathbf{x} * \mathbf{y}$)
- Quadratic and higher order terms (\mathbf{x}^2 , \mathbf{x}^3 , etc.)
- Exponents
- Logarithms
- Absolute values

A **linear constraint** is expressed by an equality or inequality as follows:

- Linear expression == Linear expression
- Linear expression >= Linear expression
- Linear expression <= Linear expression.

Any linear constraint can be rewritten as one or two expressions of the type:

- Linear expression <= 0

Continuous variables are variables that take their values from the set of real numbers. Restrictions on their values that create discontinuities, for example a restriction that a variable should take integer values, are not allowed.

Symbolic representation of an LP

A typical symbolic representation of an LP is as follows:

$$\text{minimize } c(0) + c(1)x(1) + \dots + c(n)x(n)$$

subject to

$$a(11)x(1) + a(12)x(2) \dots + a(1n)x(n) \geq b(1)$$

$$a(21)x(1) + a(22)x(2) \dots + a(2n)x(n) \geq b(2)$$

...

$$a(m1)x(1) + a(m2)x(2) \dots + a(mn)x(n) \geq b(m)$$

$$x(1), x(2), \dots, x(n) \geq 0$$

This can be written in a shorter form as:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

The first line is the objective function, namely to minimize a linear function of the decision variables, \mathbf{x} . \mathbf{c} is the vector of objective coefficients.

The second line in the short form represents the constraints. \mathbf{A} is the matrix of constraint coefficients, \mathbf{x} is the vector of decision variables, and \mathbf{b} is the vector of right-hand side coefficients.

The third line represents the upper and lower bounds on the variables. \mathbf{lb} is the vector of lower bound constants, and \mathbf{ub} is the vector of upper bound constants.

Characteristics of a linear program

The following table summarizes the characteristics of the LP model elements, where \mathbf{x} , \mathbf{y} and \mathbf{z} represent decision variables.

Model element	Characteristic	Examples	
		Acceptable	Not acceptable
Objective function	Linear	minimize $(2*x+5*y)$	minimize $(2*x*y +5*y)$
		maximize $(6*x+7*y+8*z)$	maximize $(2*x^2 +\log(y))$
Constraints	Linear	$2*x+5*y \leq 7$	$2*x*y +5*y \leq 7$
	Inequalities must be of the form \geq or \leq . No strict inequalities ($<$ or $>$) are allowed.	$2*x+5*y \leq 7$	$2*x+5*y < 7$
Decision variables	Continuous	$x \in [-1,1]$ (x can take any real value from -1 to 1)	$x \in \{0,1\}$ (x must be either 0 or 1)

Example: a production problem

Learning objective

Analyze a simple production problem in terms of decision variables, an objective function, and constraints. Write an LP formulation of this problem, and learn how to construct a graphical representation of the model. Understand what feasible, optimal, infeasible and unbounded mean in the context of LP.

Key terms

- decision variable
- objective function
- constraint
- feasible
- optimal
- infeasible
- unbounded

In this topic you will consider a simple Linear Programming (LP) problem, namely a telephone production problem, and develop a descriptive model for it. For people familiar with computer programming methodology, writing a descriptive model is a lot like writing pseudo-code when developing applications. Basically, you write a problem description in words in terms of the decision variables, objective function and constraints prior to writing it in mathematics and using a specialized language such as OPL.

You'll see how to convert the description model into a mathematical model, and how this model can be represented graphically.

Problem description: Telephone production

A telephone company produces and sells two kinds of telephones:

- Desk phones
- Cellular phones

Each type of phone is assembled and painted by the company. The objective is to maximize profit and the company has to produce at least 100 of each type of phone. However, there are limits in

terms of the company's production capacity. In order to do this, the company has to calculate the optimal number of each type of phone to produce, while not exceeding the capacity of the plant.

The production data is as follows:

- A desk phone's processing time is:
 - 12 min. on the assembly machine
 - 30 min. on the painting machine
- A cellular phone's processing time is:
 - 24 min. on the assembly machine
 - 24 min. on the painting machine
- The assembly machine is available for at most 400 hours.
- The painting machine is available for at most 490 hours.
- Desk phones return a profit of \$12 per unit.
- Cellular phones return a profit of \$20 per unit.

Practice



Write a descriptive model

Write a descriptive model of the telephone production problem. Don't look at the solution (which follows) until you have attempted this yourself. To do this, answer the following questions:

- What are the decision variables? Write down names for each decision variable.
- What is the objective?
- What are the constraints?

Telephone production: descriptive model

A possible descriptive model of the telephone production problem is as follows (you may, for example, have chosen different names for your decision variables, but the basic elements should be the same):

- Decision variables:
 - Number of desk phones produced (DeskProduction)
 - Number of cellular phones produced (CellProduction)
- Objective: Maximize profit
- Constraints:
 - The DeskProduction should be greater than or equal to 100.
 - The CellProduction should be greater than or equal to 100.
 - The assembly time for DeskProduction plus the assembly time for CellProduction should not exceed 400 hours.
 - The painting time for DeskProduction plus the painting time for CellProduction should not exceed 490 hours.

Write a mathematical model

Next, convert your descriptive model into a mathematical model using the two decision variables DeskProduction and CellProduction. To do this, take the objective and each constraint in turn, and write them as mathematical expressions. For the constraints, use either “=”, “<=”, or “>=”. Also write down the domain for the decision variables.

Write your model in the typical format of optimization models, namely:

maximize Objective

subject to Constraints

Variable domains

Use the data given in the problem description, and remember to convert minutes to hours where appropriate. You can give names to your constraints to describe their purpose.

Telephone production: mathematical model

The mathematical formulation of the telephone production problem is as follows:

maximize $12 * \text{DeskProduction} + 20 * \text{CellProduction}$

subject to

DeskProductionLimit: $\text{DeskProduction} \geq 100$

CellProductionLimit: $\text{CellProduction} \geq 100$

AssemblyTimeLimit: $0.2 * \text{DeskProduction} + 0.4 * \text{CellProduction} \leq 400$

PaintingTimeLimit: $0.5 * \text{DeskProduction} + 0.4 * \text{CellProduction} \leq 490$

DeskProduction, CellProduction are real numbers

This problem is a typical example of an LP model. Real-world models will typically involve many more variables and constraints, or even additional objectives. In general, when formulating an optimization model, it's important to keep the model as simple as possible: Only include those decision variables, objectives and constraints that are absolutely essential for solving the model. You may end up going through a few iterations before finding the right model, as you discover essential constraints that you haven't accounted for, or perhaps you've included constraints that end up being irrelevant.

Save your model if you are also taking the 3-day **Optimization with IBM ILOG OPL** course, where you'll use it again.

Graphical representation

A simple 2-dimensional LP (that is, with 2 decision variables), such as the telephone production LP, can be represented graphically using a y- and x-axis. This is often done to demonstrate optimization concepts as you'll see later in this lesson. To do this, follow these steps:

- Assign one variable to the x-axis and the other to the y-axis.
- Draw each of the constraints as you would draw any line in 2 dimensions.
- Use the signs of the constraints ($=$, \leq or \geq) to determine which side of each line falls within the feasible region (allowable solutions).
- Draw the objective function as you would draw any line in 2 dimensions, by substituting any value for the objective (for example, $12 * \text{DeskProduction} + 20 * \text{CellProduction} = 4000$ where we've substituted the value of 4000 to be able to draw a line).

To find the optimal solution, move the objective line either up (for maximizing) or down (for minimizing) until the furthest point in the feasible region is reached. This furthest point (or line) then represents the optimal solution(s).

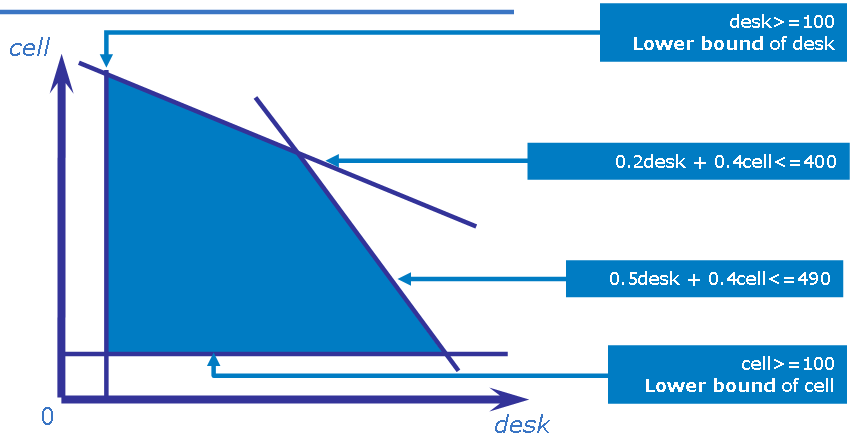
You can do this for the telephone production problem by assigning CellProduction to the y-axis, and DeskProduction to the x-axis.

Feasible region

Recall that the feasible region of an LP is the region delimited by the constraints, and it represents all feasible solutions.

The feasible region for the telephone production problem looks as follows (DeskProduction and CellProduction are abbreviated to be “desk” and “cell” instead).

Feasible set of solutions



Practice

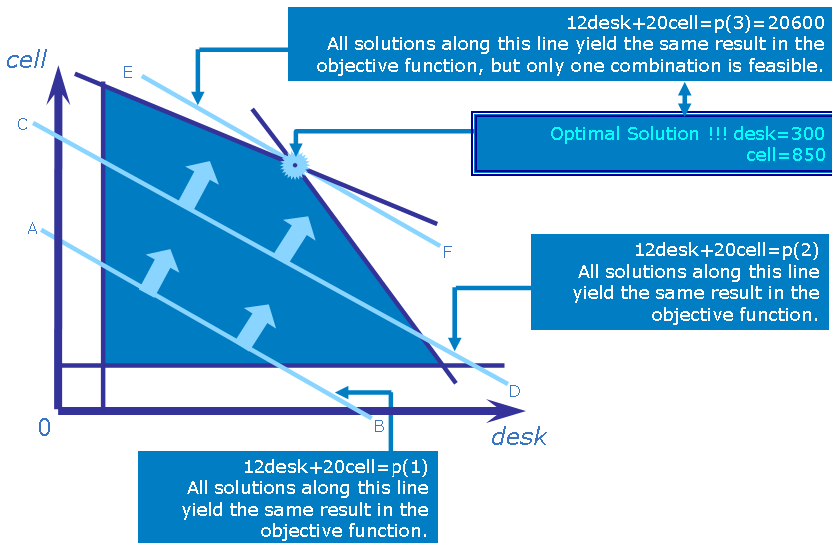


Look at the diagram and search, intuitively, for the optimal solution, without drawing the objective function. Next, you'll learn how to find the optimal solution graphically by drawing lines representing the objective function.

Finding the optimal solution

To find the optimal solution to the LP, you must find values for the decision variables, within the feasible region, that maximize profit as defined by the objective function (here, $12 * \text{desk} + 20 * \text{cell}$). To do this, first draw a line representing the objective by substituting a value for the objective. Next move the line up (because this is a maximization problem) to find the point where the line last touches the feasible region. This is shown in the following diagram:

The optimal solution



All the solutions that yield a given result of the objective function (for example, $p(1)$) are found along a single line (linear relationship), in this case, line AB. Other values of the objective function (for example, $p(2)$) will be found along parallel lines (that is, lines with the same slope, as per line CD). In a profit maximizing problem such as this one, these parallel lines are often called **isoprofit** lines, because all the points along such a line represent the same profit. In a cost minimization problem, they are known as **isocost** lines.

Since all isoprofit lines have the same slope, you can find all other isoprofit lines by pushing the objective value further out, moving in parallel, until the isoprofit lines no longer intersect the feasible region. The last isoprofit line that touches the feasible region defines the largest (therefore maximum) possible value of the objective function. In the case of the telephone production problem, this is found along line EF.



The optimal solution of a linear program always belongs to an extreme point of the feasible region (that is, at a vertex or an edge).

Multiple optimal solutions

It is possible that an LP has multiple optimal solutions.

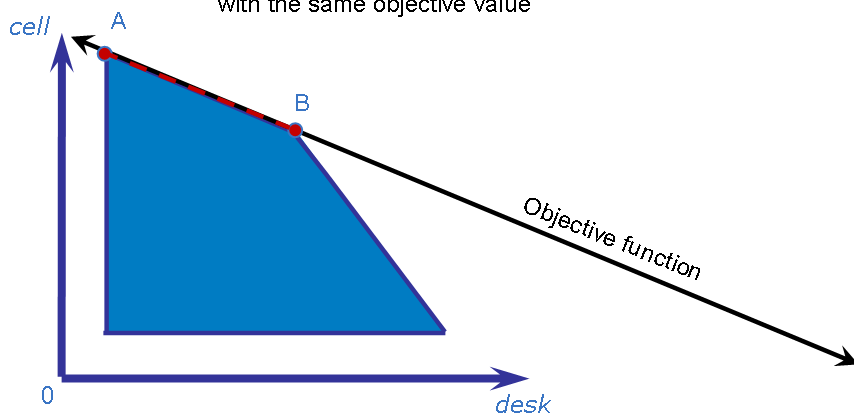
At least one optimal solution will be at a vertex.

By default, CPLEX reports the first optimal solution found.

Example of multiple optimal solutions

Slope of objective function = slope of constraint on line AB

All the points on line AB are optimal solutions with the same objective value



Binding and nonbinding constraints

A constraint is **binding** if the constraint becomes an equality when the solution values are substituted.

Graphically, binding constraints are constraints where the optimal solution lies exactly on the line representing that constraint.

In the telephone production problem, the constraint limiting time on the assembly machine is binding:

$$\begin{aligned}
 0.2\text{desk} + 0.4 \text{ cell} &\leq 400 \\
 \text{desk} &= 300 \\
 \text{cell} &= 850 \\
 0.2(300) + 0.4(850) &= 400
 \end{aligned}$$

The same is true for the time limit on the painting machine:

$$\begin{aligned}
 0.5\text{desk} + 0.4\text{cell} &\leq 490 \\
 0.5(300) + 0.4(850) &= 490
 \end{aligned}$$

On the other hand, the requirement that at least 100 of each telephone type be produced is **nonbinding** because the left and right hand sides are not equal:

`desk >= 100`

`desk = 300`

`300 ≠ 100`

`cell >= 100`

`cell = 850`

`850 ≠ 100`

In fact, the optimal production values are quite far from their lower bounds. However, such constraints may become binding if the model were revised with different data or additional constraints and it is therefore important to keep these constraints in the model.

Infeasibility

A model is infeasible when no solution exists that satisfies all the constraints. This may be because:

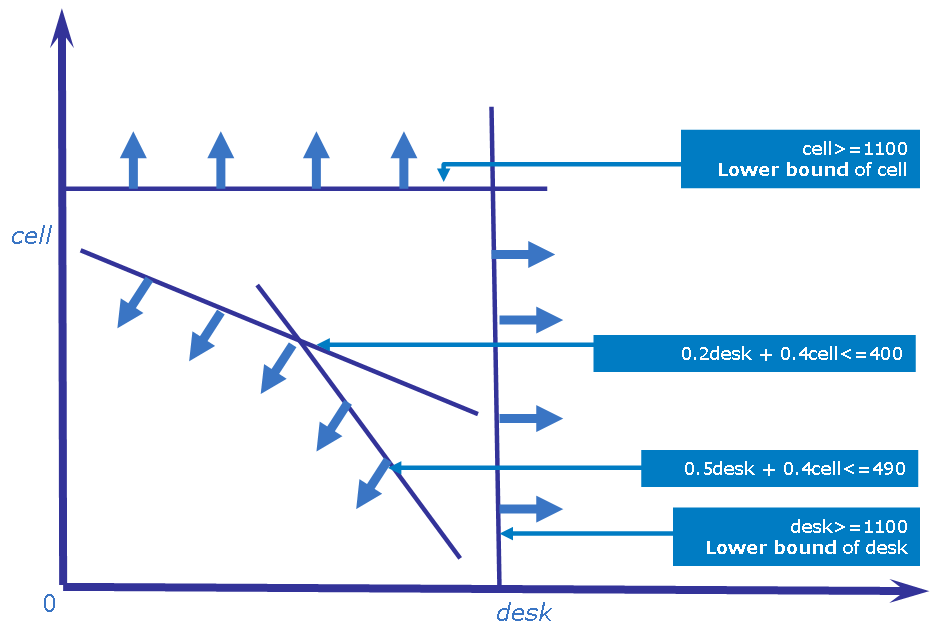
- The model formulation is incorrect.
- The data is incorrect.
- The model and data are correct, but represent a real-world conflict in the system being modeled.

When faced with an infeasible model, it's not always easy to identify the source of the infeasibility. OPL provides functionality to help you identify potential causes of infeasibilities, and it will also suggest changes to make the model feasible.

In the telephone production problem, the model could become infeasible for several reasons. For example, if someone doing the data entry had accidentally typed in lower bounds on production of 1100 instead of 100, there will be no possible solution.

The following graphic represents this situation where the arrow directions show in which direction the constraints are valid. The valid regions do not intersect and there is no possible feasible solution. The feasible region is empty.

Infeasible set of constraints



Correcting infeasible models

To correct an infeasible model, you must use your knowledge of the real-world situation you are modeling.

If you know that the model is realizable, you can usually manually construct an example of a feasible solution and use it to determine where your model or data is incorrect. For example, the telephone production manager may input the previous month's production figures as a solution to the model and thereby discover that they violate the erroneously entered bounds of 1100.

OPL can help perform infeasibility analysis, which can get very complicated in large models.

OPL may suggest relaxing one or more constraints.

In the case of LP models, the term “relaxation” refers to changing the right hand side of the constraint to allow some violation of the original constraint.

For example, a relaxation of the assembly time constraint is as follows:

$$0.2 * \text{desk} + 0.4 * \text{cell} \leq 440$$

Here, the right hand side has been relaxed from 400 to 440, meaning that you allow more time for assembly than originally planned. This may be a realistic scenario if, for example, overtime is available. But how can one represent the difference between, for example, regular time and overtime mathematically? The answer is to use soft constraints.

Converting hard constraints to soft constraints

A **soft constraint** is a constraint that can be violated in some circumstances. Conversely, a **hard constraint** cannot be violated under any circumstances.

Converting hard constraints to soft is one way to resolve infeasibilities.

For example, the original hard constraint on assembly time is as follows:

$$0.2 * \text{desk} + 0.4 * \text{cell} \leq 400$$

You can turn this into a soft constraint if you know that, for example, an additional 40 hours of overtime are available at an additional cost. To do this, first add an overtime term to the righthand side:

$$0.2 * \text{desk} + 0.4 * \text{cell} \leq 400 + \text{overtime}$$

Next, add a hard limit to the amount of overtime available:

$$\text{overtime} \leq 40$$

Finally, add an additional cost to the objective to penalize use of overtime. Assume that in this case overtime costs an additional \$2/hour, then the new objective becomes:

$$\text{Maximize } 12 * \text{desk} + 20 * \text{cell} - 2 * \text{overtime}$$

Maximizing a negative number is the same as minimizing a positive number. You are therefore effectively telling the model to minimize the amount of overtime used, but that it can be used if absolutely necessary.

Unbounded variable versus unbounded model

A *variable* is **unbounded** when one or both of its bounds is infinite.

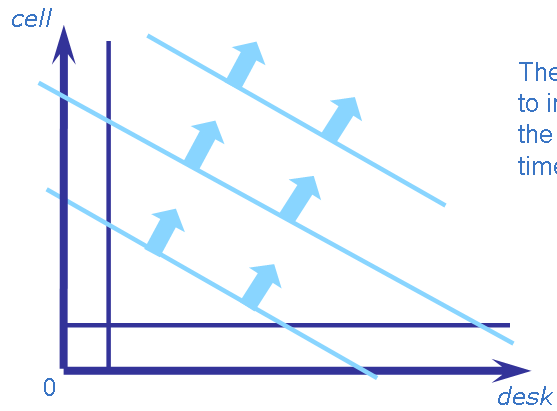
A *model* is unbounded when its objective value can be increased or decreased without limit.

The fact that a variable is unbounded does not necessarily influence the solvability of the model and should not be confused with a model being unbounded.

An unbounded model is almost certainly not correctly formulated. While an infeasibility implies a model where constraints that are too limiting, unboundedness implies a model where an important constraint is either missing or not restrictive enough.

The telephone production problem will become unbounded if, for example, the constraints on the assembly and painting time were neglected. The feasible region would then look as follows and as you can see the objective value can increase without limit, up to infinity, seeing that there is no boundary to the region.

Unbounded feasible region



Algorithms for solving LP problems

Learning objective

Gain familiarity with some of the techniques available in OPL to solve optimization problems.

Key terms

- simplex algorithm
- primal problem
- dual problem
- dual simplex algorithm
- barrier algorithm
- reduced costs
- slacks

In this topic you'll gain familiarity with some of the algorithms used by the CPLEX engine to solve LP problems in OPL. These algorithms include:

- the simplex algorithm
- the dual simplex algorithm
- the barrier algorithm

The simplex algorithm

The telephone production problem is a simple example with only two decision variables. Real-world optimization problems typically have many more decision variables, and models with millions of constraints and variables are not unusual.

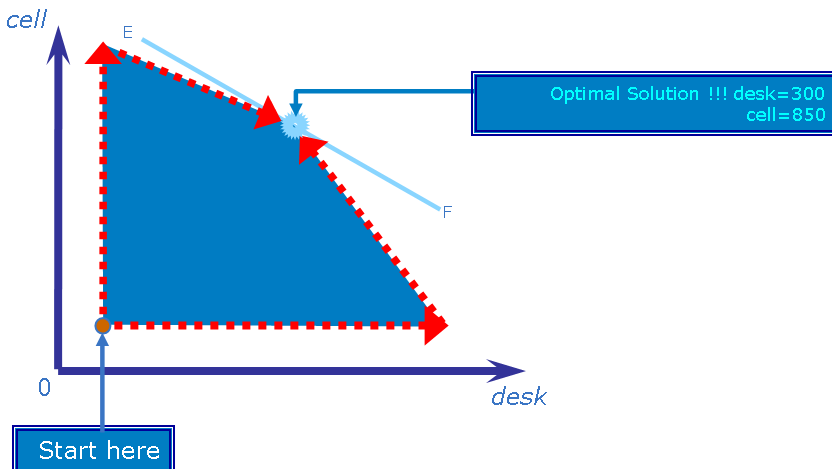
A graphical solution such as the one followed with the telephone problem is not possible for larger models.

The simplex algorithm, developed by George Dantzig in 1947, was the first generalized algorithm for solving LP problems, and is the basis for many optimization algorithms.

The simplex method is an iterative method. It starts with an initial feasible solution, and then tests to see if it can improve the result of the objective function. It continues until the objective function cannot be further improved.

In the graphic example used for the telephone production problem, this is the equivalent of starting somewhere along the edge of the shaded feasible region, and advancing vertex-by-vertex until arriving at the vertex that also intersects the isoprofit line EF.

The Simplex algorithm: finding the optimal solution



The revised simplex algorithm

In order to make it run faster and more efficiently on computers, George Dantzig and W. Orchard-Hays revised the simplex algorithm in 1953.

IBM® ILOG CPLEX uses the revised simplex algorithm and includes a number of other improvements, providing a particularly efficient implementation that can solve very large problems rapidly.

You can tune some CPLEX parameters to change the algorithmic behavior according to your needs.

The dual of an LP

Every LP problem has an associated LP problem known as its **dual**.

The dual of this associated problem is the original LP problem (known as the **primal** problem).

If the primal problem is a minimization problem, then the dual problem is a maximization problem and vice versa.

Next, you'll learn more about:

- what a dual formulation looks like
- the significance of the dual variables
- the dual-simplex algorithm

A generic example of a primal problem and its associated dual is shown next. There are standard rules for deriving the dual from the primal, but this is beyond the scope of this training.

A primal-dual pair

A primal-dual pair

Primal (P):

$$\max z = c_1x_1 + c_2x_2 + \dots + c_nx_n$$

s.t.

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m$$

$$x_j \geq 0 \quad (j = 1, 2, \dots, n)$$

Dual (D):

$$\min w = b_1y_1 + b_2y_2 + \dots + b_my_m$$

s.t.

$$a_{11}y_1 + a_{21}y_2 + \dots + a_{m1}y_m \geq c_1$$

$$a_{12}y_1 + a_{22}y_2 + \dots + a_{m2}y_m \geq c_2$$

$$\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots$$

$$a_{1n}y_1 + a_{2n}y_2 + \dots + a_{mn}y_m \geq c_n$$

$$y_i \geq 0 \quad (i = 1, 2, \dots, m)$$

- Each constraint in the primal has an associated **dual variable**, y_i .
- Any feasible solution to D is an upper bound to P, and any feasible solution to P is a lower bound to D.
- In LP, the **optimal objective** values of D and P are equivalent, and **occurs where these bounds meet**.
- The **dual** can help solve difficult primal problems by providing a **bound** that in the **best case equals the optimal solution to the primal problem**.

Lesson 3: Beyond Simple LP / Topic: Network Models



© Copyright IBM Corporation 2009

1

Dual prices

In any solution to the dual, the values of the dual variables are known as the dual prices (also called shadow prices).

For each constraint in the primal problem, its associated dual price indicates how much the dual objective will change with a unit change in the right hand side of the constraint.

The dual price of a non-binding constraint is zero. That is, changing the right hand side of the constraint will not affect the objective value.

The dual price of a binding constraint can help you make decisions regarding the constraint.

For example, the dual price of a binding resource constraint can be used to determine whether more of the resource should be purchased or not.

The dual simplex algorithm

The simplex algorithm works by finding a feasible solution and moving progressively toward optimality.

The dual simplex algorithm implicitly uses the dual to try and find an optimal solution to the primal as early as it can, and regardless of whether the solution is feasible or not.

It then moves from one vertex to another, gradually decreasing the infeasibility while maintaining optimality, until an optimal feasible solution to the primal problem is found.

The dual simplex algorithm is the first choice for most LP problems in OPL.

Basic solutions and basic variables

You learned earlier that the simplex algorithm travels from vertex to vertex to search for the optimal solution.

A solution at a vertex is known as a basic solution.

Without getting into detail, it's worth knowing that part of the simplex algorithm involves setting a subset of variables to zero at each iteration. These variables are known as non-basic variables. The remaining variables are the basic variables.

The concepts of basic solutions and variables are relevant in the definition of reduced costs that follows next.

Reduced costs

The reduced cost of a variable gives an indication of the amount the objective will change with a unit increase in the variable value.

Consider the simplest form of an LP:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s. t.} \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

If \mathbf{y} represents the dual variables for a given basic solution, then the reduced costs are defined as:

$$\mathbf{c} - \mathbf{y}^T \mathbf{A}$$

Such a basic solution is optimal if

$$\mathbf{c} - \mathbf{y}^T \mathbf{A} \geq \mathbf{0}$$

If all reduced costs for this LP are non-negative, it follows that the objective value can only increase with a change in the variable value, and therefore the solution (when minimizing) is optimal.

Default CPLEX optimality criteria

Because CPLEX operates on finite precision computers, it uses an optimality tolerance to test the reduced costs.

The default optimality tolerance is $-1\text{e-}6$, with optimality criteria for the simplest form of an LP then being:

$$\mathbf{c} - \mathbf{y}^T \mathbf{A} > -1\text{e-}6$$

You can adjust this optimality tolerance, for example if the algorithm takes very long to converge and has already achieved a solution sufficiently close to optimality.

The CPLEX optimality test for other forms of LPs, for example LPs with finite lower and upper bounds on the variables, becomes a little more complex and is not covered as part of this course. CPLEX handles these tests automatically and the user can control the optimality tolerance.

Reduced costs and multiple optimal solutions

In the earlier example you saw how one can visualize multiple optimal solutions for an LP with two variables.

For larger LPs, the reduced costs can be used to determine whether multiple optimal solutions exist.

Multiple optimal solutions exist when one or more non-basic variables with a zero reduced cost exist in an optimal solution (that is, variable values that can change without affecting the objective value).

In order to determine whether multiple optimal solutions exist, you can examine the values of the reduced costs in OPL.

Slack values

For any solution, the difference between the left and right hand sides of a constraint is known as the **slack** value for that constraint.

For example, if a constraint states that $f(\mathbf{x}) \leq 100$, and in the solution $f(\mathbf{x}) = 80$, then the slack value of this constraint is 20.

In the earlier example, you learned about binding and non-binding constraints. For example, $f(\mathbf{x}) \leq 100$ is binding if $f(\mathbf{x}) = 100$, and non-binding if $f(\mathbf{x}) = 80$.

The slack value for a binding constraint is always zero, that is, the constraint is met exactly.

You can determine which constraints are binding in a solution by examining the slack values in OPL.

This might help to better interpret the solution and help suggest which constraints may benefit from a change in bounds or a change into a soft constraint.

Degeneracy

It is possible that multiple non-optimal solutions with the same objective value exist.

As the simplex algorithm attempts to move in the direction of an improved objective value, it might happen that the algorithm starts cycling between non-optimal solutions with equivalent objective values. This is known as degeneracy.

Modern LP solvers, such as CPLEX, have built-in mechanisms to help escape such cycling by using perturbing techniques involving the variable bounds.

If the default algorithm does not break the degenerate cycle, it's a good idea to try some other algorithms, for example the dual-simplex algorithm. Problems that are primal degenerate, are often not dual degenerate, and vice versa.

CPLEX users can change the algorithm by editing CPLEX's LP method parameter.

Barrier methods

Most of the algorithms available through CPLEX call upon the basic simplex method or some variation of it.

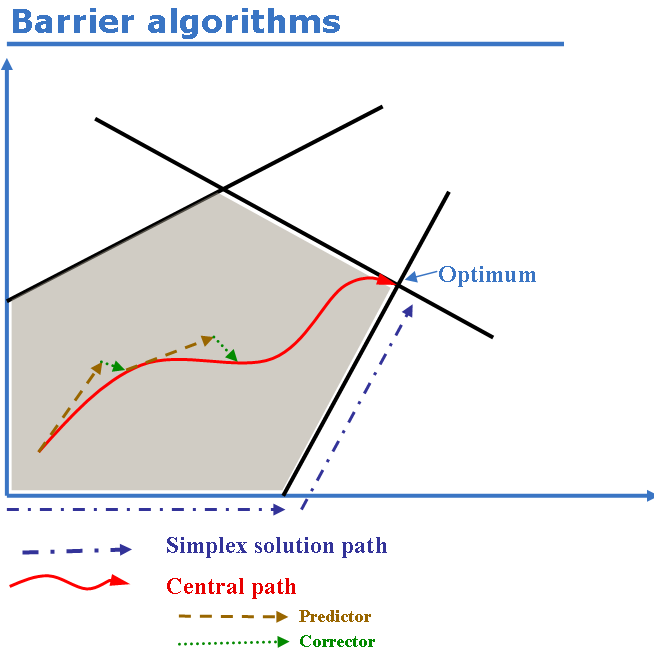
There are, however, algorithms available through CPLEX, such as the **barrier algorithm**, that use alternative methods.

In graphical terms, the simplex algorithm starts along the edge of the feasible region and searches for an optimal vertex.

The barrier method starts somewhere inside the feasible region – in other words, it avoids the “barrier” that is created by the constraints, and burrows through the feasible region to find the optimal solution.

In its search, the method uses what is known as a **predictor-corrector algorithm** that constantly adjusts its path through the center of the feasible region (the **central path**).

The following diagram illustrates how the barrier method works in comparison to the primal simplex algorithm:



More details on how and when to choose alternative optimization algorithms are given in the 3-day OPL training course. However, it's a good idea to try alternative algorithms to improve performance.

Presolve

CPLEX provides a **presolve** procedure.

Presolve evaluates the model formulation before solving it, and attempts to reduce the size of the problem that is sent to the solver engine.

A reduction in problem size typically translates to a reduction in total run time.

For example, a real problem presented to CPLEX with approximately 160,000 constraints and 596,000 decision variables, was reduced by presolve to a problem with 27,000 constraints and 150,000 decision variables.

The presolve time was only 1.32 seconds and reduced the solution time from nearly half an hour to under 25 seconds.

An example of presolve operations:

Maximize:

$$[1] \quad 2x(1) + 3x(2) - x(3) - x(4)$$

subject to:

$$[2] \quad x(1) + x(2) + x(3) - 2x(4) \leq 4$$

$$[3] \quad -x(1) - x(2) + x(3) - x(4) \leq 1$$

$$[4] \quad x(1) \qquad \qquad \qquad + x(4) \leq 3$$

$$[5] \quad x(1), x(2), x(3), x(4) \geq 0$$

- Because $x(3)$ has a negative coefficient in the objective, the optimization will minimize $x(3)$.
- In constraints [2] and [3] $x(3)$ has positive coefficients, and the constraints are \leq . Thus, $x(3)$ can be reduced to 0, and becomes redundant.
- In constraint [3], all the coefficients are now negative. Because the left hand side of [3] can never be positive, any assignment of values will satisfy the constraint. The constraint is redundant and can be removed.

Presolve is also available for other types of mathematical programming problems solved with CPLEX, such as Mixed-Integer Programs (MIPs) and Quadratic Programs (QPs).

Summary

Review

Linear programming (LP) deals with the maximization (or minimization) of a linear expression, known as the objective function, subject to linear constraints. LPs contain only continuous decision variables.

The feasible region of an LP is the region bounded by the constraints. Any feasible solution lies within the feasible region.

The optimal solution is a solution that lies within the feasible region (it satisfies all constraints) and also results in the best possible value of the objective function. The optimal solution always lies at an extreme point of the feasible region (on the edge) and is most often found at a vertex. It is possible to have multiple alternative optimal solutions, but at least one optimal solution will always be at a vertex.

An infeasible solution occurs when no solution exists that can satisfy all the constraints.

An unbounded model is a model that is not properly described and therefore allows the objective to increase or decrease to plus or minus infinity.

A constraint is said to be binding if the left hand side and the right hand side of the constraint are equal when the optimal values of the decision variables are inserted.

A constraint that cannot be violated under any circumstances is known as a hard constraint. A constraint that can be violated at a certain cost, is a soft constraint. Converting hard constraints to soft is one way to resolve infeasibilities.

Each LP problem has an associated LP problem that is known as the dual of the original problem (also known as the primal). When minimizing the primal, any feasible solution provides an upper bound to the dual, while any feasible solution to the dual provides a lower bound to the primal. The opposite is true when maximizing. The primal and dual problems have the same optimal value, namely the value where these bounds meet.

Some of the algorithms used by the CPLEX engine to solve LP problems in OPL include:

- the simplex algorithm
- the dual simplex algorithm
- the barrier algorithm

CPLEX includes a Presolve function to reduce the size of the LP sent to the optimization engine.

This page is intentionally left blank.

Lesson 3: Beyond Simple LP

At the end of this lesson you will be able to:

- Describe what a network model is, and the benefits of using network models
- Understand the concepts of nonlinearity and convexity
- Describe what a piecewise linear function is
- Describe the differences between:
 - Linear Programming (LP)
 - Integer Programming (IP)
 - Mixed-Integer Programming (MIP)
 - Quadratic Programming (QP)
- Construct a simple MIP model

Beyond LP

In this lesson, you'll learn about some special cases of LP, as well as some other non-LP techniques, and learn under which conditions they should be used:

- Network models
- Piece-wise linear models
- Integer programming
- Mixed-integer programming
- Quadratic programming

Many more mathematical programming topics exist beyond what is covered in this lesson. Recommended reading includes:

- *Model Building in Mathematical Programming* by H. Paul Williams
- *Operations Research: Applications and Algorithms* by Wayne L. Winston

Network models

Learning objective

Understand what a network model is and how its structure can be exploited.

Key terms

- network
- node
- arc

Networks in real life

Several problems encountered in Operations Research (OR) involve networks, such as:

- Distribution problems (for example, transportation networks)
- Assignment problems (for example, networks of workers and jobs they could be assigned to)
- Planning problems (for example, critical path analysis for project planning)

Many network models are special LP problems for which specialized solution algorithms exist.

It is important to know whether a problem can be formulated as a network model to exploit the special structure.

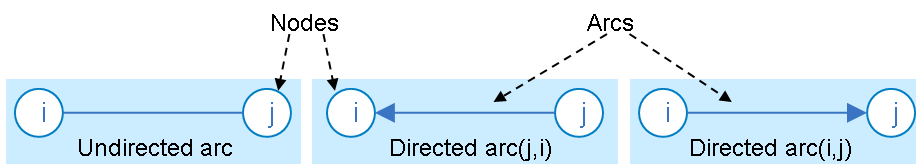
This topic introduces networks in general, as well as some well-known instances of network models.

Network modeling concepts (1)

Any network structure can be described using two types of objects:

- **Nodes:** Defined points in the network, for example warehouses.
- **Arcs:** An arc connects two nodes, for example a road connecting two warehouses.

An arc can be directed, which means that an arc $a(i,j)$ from node i to node j is different from arc $a(j,i)$ that begins at node j and ends at node i .



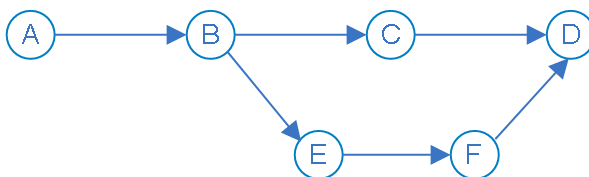
Network modeling concepts (2)

A sequence of arcs connecting two nodes is called a *chain*. Each arc in a chain shares exactly one node with the preceding arc.

When all the arcs in a chain are directed such that it is possible to traverse the chain in the directions of the arcs from the start node to the end node, it is called a *path*.



A chain from A to D



Two paths from A to D:

- 1) ABCD
- 2) ABEFD

I: 45
S: 45

Lesson 3: Beyond Simple LP / Topic: Network models

© Copyright IBM Corporation 2009

1

Types of network problems

The following are some well-known types of network problems:

- Transportation problem
- Transshipment problem
- Assignment problem
- Shortest path problem
- Critical path analysis

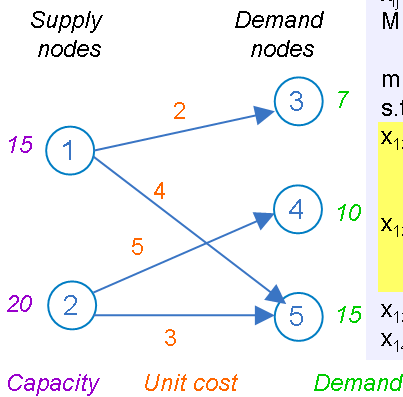
Next, you'll learn how to recognize each of these, and how their special structure can be exploited.

The Transportation Problem

The Transportation Problem

This type of problem involves a set of **supply nodes** and a set of **demand nodes**.

The objective is to **minimize the transportation cost** from the supply nodes to the demand nodes so as to **satisfy demand**, without **exceeding the suppliers' capacities**.



LP formulation

x_{ij} = quantity transported from i to j
 M = a large number for disallowed arcs

min $2x_{13} + 0x_{14} + 4x_{15} + 0x_{23} + 5x_{24} + 3x_{25}$
 s.t.

$x_{13} + x_{14} + x_{15}$	≤ 15	} out \leq supply	
$x_{23} + x_{24} + x_{25}$	≤ 20		
x_{13}	$+ x_{23}$	$= 7$	} in = demand
x_{14}	$+ x_{24}$	$= 10$	
x_{15}	$+ x_{25}$	$= 15$	

$x_{13}, x_{15}, x_{24}, x_{25} \geq 0$ (flow conservation constraints)
 $x_{14} = x_{23} = 0$

I: 45
 S: 45

Lesson 3: Beyond Simple LP / Topic: Network models

© Copyright IBM Corporation 2009

1

Arcs (1,4) and (2,3) do not exist, but the variables x_{14} and x_{23} are included here to illustrate the special structure of the transportation problem. However, if you were to formulate such a model in practice, you would exclude these variables.

The special structure of the transportation problem, as well as many other types of network problems, allow the use of specialized algorithms that lead to significant reductions in solution time.

Another important characteristic of transportation problems (and also some other network problems) is that if all the capacities and demands are integer, then the decision variables will take integer values.

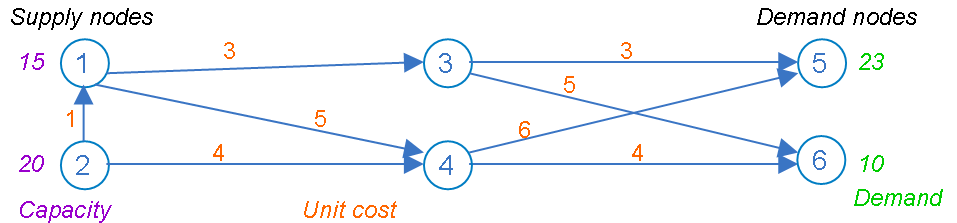
This is important to know, because it means that you do not have to use integer variables in such cases.

As you'll learn in a later topic, integer variables often lead to problems that require much more computational effort compared to problems with only continuous variables.

The Transshipment Problem

The Transshipment Problem

This problem is similar to the transportation problem, except that intermediate nodes exist in addition to supply and demand nodes.



LP formulation

$$\min 1x_{21} + 3x_{13} + 5x_{14} + 4x_{24} + 3x_{35} + 5x_{36} + 6x_{43} + 4x_{45} + 4x_{46}$$

$$\text{s.t. } \left. \begin{array}{l} \text{node 1: } x_{13} + x_{14} \leq 15 + x_{21} \\ \text{node 2: } x_{21} + x_{24} \leq 20 \\ \text{node 3: } x_{13} = x_{35} + x_{36} \\ \text{node 4: } x_{14} + x_{24} = x_{43} + x_{45} + x_{46} \\ \text{node 5: } x_{35} + x_{45} = 23 \\ \text{node 6: } x_{36} + x_{46} = 10 \end{array} \right\} \begin{array}{l} \text{out} \leq \text{capacity} + \text{in} \\ \\ \text{in} = \text{out} \\ \text{in} = \text{demand} \end{array} \quad \text{(flow conservation constraints)}$$

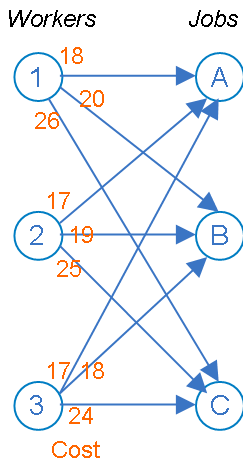
$$x_{ij} \geq 0 \text{ for all } i, j$$

It is possible to write the transshipment problem as a transportation problem in order to use specialized algorithms that exploit the structure of transportation problem. This conversion is not covered as part of this course (CPLEX does this conversion automatically), but you can find details in the textbooks listed at the end of this course.

The Assignment Problem

The Assignment Problem

This is the problem of assigning one set of items to another, while optimizing a given objective. For example, the problem of assigning workers to jobs so as to minimize the hiring costs.



Formulation

$$\min 18x_{1A} + 20x_{1B} + 26x_{1C} + 17x_{2A} + 19x_{2B} + 25x_{2C} + 17x_{3A} + 18x_{3B} + 24x_{3C}$$

$$\text{s.t. } \begin{cases} x_{1A} + x_{2A} + x_{3A} = 1 \\ x_{1B} + x_{2B} + x_{3B} = 1 \\ x_{1C} + x_{2C} + x_{3C} = 1 \end{cases} \left. \begin{array}{l} \text{Each job is filled} \\ \text{by exactly one person} \end{array} \right\} \text{(flow conservation constraints)}$$

$$\begin{cases} x_{1A} + x_{1B} + x_{1C} = 1 \\ x_{2A} + x_{2B} + x_{2C} = 1 \\ x_{3A} + x_{3B} + x_{3C} = 1 \end{cases} \left. \begin{array}{l} \text{Each person is assigned} \\ \text{to exactly one job} \end{array} \right\}$$

$x_{ij} = 1$ if worker i assigned to job j , 0 otherwise

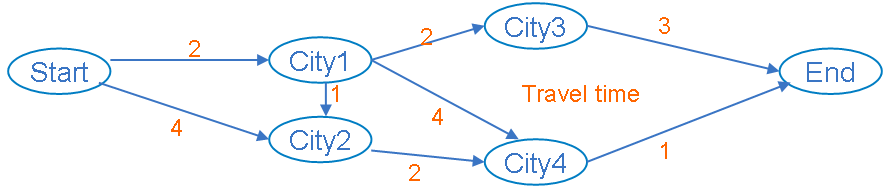
This is a special case of the transportation problem, with all supply capacity = 1 and all demand = 1.

Even though x_{ij} must take 0-1 values, it can be declared continuous due to the integrality property.

The Shortest Path Problem

The Shortest Path Problem

This is the problem of finding the shortest path through a network. For example, find the minimum travel time between two cities in a network of cities.



Formulation

$$\min 2x_{S1} + 4x_{S2} + 2x_{13} + 4x_{14} + 1x_{12} + 2x_{24} + 3x_{3E} + 1x_{4E}$$

s.t.

$$\text{Start city: } x_{S1} + x_{S2} = 1$$

$$\text{City 1: } x_{S1} = x_{12} + x_{13} + x_{14}$$

$$\text{City 2: } x_{S2} + x_{12} = x_{24}$$

$$\text{City 3: } x_{13} = x_{3E}$$

$$\text{City 4: } x_{14} + x_{24} = x_{4E}$$

$$\text{End city: } x_{3E} + x_{4E} = 1$$

$x_{ij} = 1$ if the arc from i to j is included in the shortest path, 0 otherwise

Exactly one arc chosen in and out of each city (flow conservation constraints)

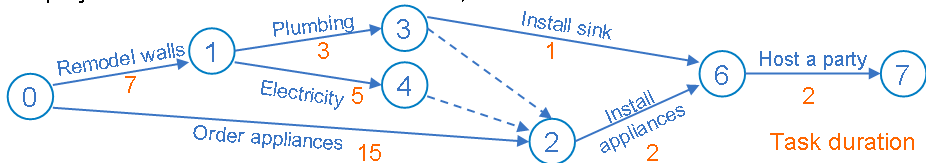
Again, even though x_{ij} must take 0-1 values, it can be declared continuous due to the integrality property.

The shortest path problem is a special case of the transshipment problem, where there is exactly one supply node and one demand node, and the supply and demand are both 1.

Critical Path Analysis

Critical Path Analysis

This technique is used in project planning to find the set of critical activities (the longest path through the network) where a delay in one of the critical activities will lead to an overall project delay. For example, consider a kitchen remodeling project where arcs show task durations, and nodes are task start times:



Formulation: Let t_i be the start time of tasks originating at node i .

The objective is to minimize the project completion time, t_7 .

s.t.

$t_1 - t_0 \geq 7$	} Constraints indicating task duration and precedence. 3-2 and 4-2 are dummy tasks.
$t_2 - t_0 \geq 15$	
$t_3 - t_1 \geq 3$	
$t_4 - t_1 \geq 5$	
$t_6 - t_3 \geq 1$	
$t_2 - t_3 \geq 0$	
$t_2 - t_4 \geq 0$	
$t_6 - t_2 \geq 2$	
$t_7 - t_6 \geq 2$	

Lesson 3: Beyond Simple LP / Topic: Network models 1

The critical path is the longest path in the network. It represents the minimum completion time of the project, and if any task on the critical path is delayed, the entire project will be delayed.

Tasks that do not lie on the critical path may be delayed to some extent without impacting the final completion date of the project. The critical path will change if a non-critical task is delayed to a point where it becomes critical.

Specialized algorithms exist for critical path analysis.

Specialized network algorithms in OPL

Many network problems are special types of LP problems. In many cases, using the primal or dual simplex algorithms is the most efficient way to solve them. In some cases, however, specialized algorithms tailored to the solution of problems with network structures can solve such problems more efficiently.

In OPL, a network algorithm is automatically invoked when it's likely that it would improve solution time compared to, for example, the simplex or dual simplex algorithm. It is also possible to force the use (or not) of the network algorithm by setting a parameter in OPL.

Nonlinearity and convexity

Learning objective

Understand the concepts of nonlinearity and convexity.

Key terms

- nonlinear
- convex
- non-convex

In many problems, the relationships between decision variables are not linear. Examples of industries where nonlinear optimization problems are often encountered are:

- The chemical process industry, for example oil refining or pipeline design
- The finance sector, for example stock portfolio optimization

Nonlinear Programming (NLP) is often used to solve such optimization problems. One of the NLP techniques available through OPL is Quadratic Programming (QP), used especially for portfolio optimization.

Sometimes it is possible to approximate a nonlinear function with a set of linear functions and then solve the problem using LP techniques. The success of this depends on whether the nonlinear function is **convex** or **nonconvex**.

Convexity

A region is **convex** if any point lying on a straight line between any two points in the region is also in the region.

Convex and nonconvex regions

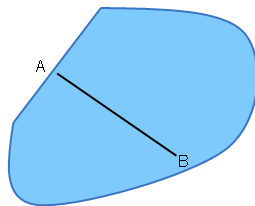


Figure 1: Convex feasible region

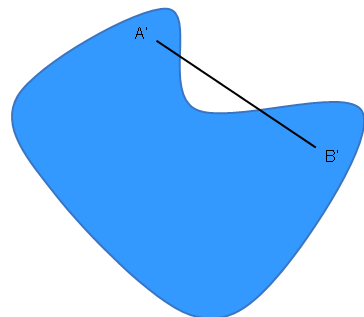


Figure 2:
Nonconvex feasible region

In the diagram above, any straight line between two points inside the region in Figure 1 (e.g. line AB) remains entirely in the region, and this region is convex. Line $A'B'$ in Figure 2, on the other hand, connects two points inside the region but passes outside it to get from one to the other. It is a nonconvex region. The concept of convexity can be extrapolated to as many dimensions as necessary.

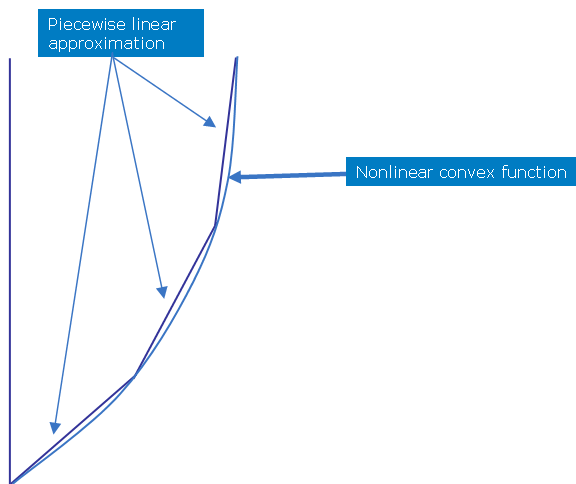
A convex feasible region is defined by a set of convex functions. A minimization problem is convex if the objective and all the constraints are convex. Minimizing a convex function, $f(x)$, is equivalent to maximizing $-f(x)$. Therefore, a maximization problem is convex if the objective is the negative of a convex function, and the feasible region is convex.

All LPs are convex, because an LP involves the minimization of a linear function over a feasible region defined by a set of linear functions, and linear functions are convex.

Approximating a nonlinear program with an LP

A convex nonlinear function can be approximated by a set of linear functions, as shown in the next figure. This could be used to allow solution with LP techniques.

Approximating nonlinearity



This approximate function consisting of several linear line segments is called a **piecewise linear** function.

Piecewise Linear Programming

Learning objective

Understand what a piecewise linear function is and how it can be used in optimization models.

Key terms

- piecewise linear function
- breakpoint
- slope

Piecewise linear programming is used when dealing with functions consisting of several linear segments. Examples of where this may be appropriate are:

- Piecewise linear approximations of convex nonlinear functions
- Piecewise linear functions representing economies of scale, for example a unit cost that changes when at least a certain number of units are ordered.

Example: Economies of scale

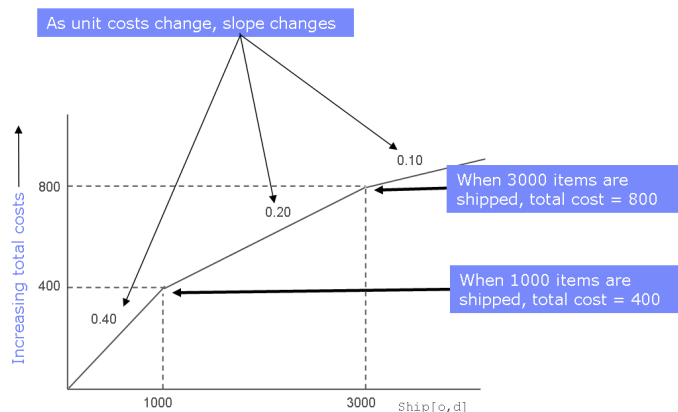
Consider a transportation problem in which the transportation cost between an origin, o , and a destination, d , depends on the size of the shipment, $ship[o][d]$:

- The first 1000 items have a shipping cost of 0.40 per item
- For the next 2000 items (i.e. items 1001-3000) the cost is 0.20 per item
- From item 3001 on, the cost decreases to 0.10 per item.

The cost of each quantity bracket remains intact (that is, the cost per unit changes only for additional units, and remains unchanged for the previous quantity bracket). Therefore, within each bracket there is a linear relationship between cost and quantity, but at each **breakpoint** the rate of linear variation changes.

If you graph this function, you see that at each breakpoint, the **slope** of the line changes. The section of the graph between two breakpoints can thus be described as a **linear piece**.

Unit costs vary with quantity



The diagram shows that the total shipping cost is evaluated by 3 different linear functions, each determined by the quantity shipped:

- $0.40 * \text{items}$ when $\text{ship}[\text{o}][\text{d}] \leq 1000$
- $0.40 * 1000 + 0.20 * (\text{ship}[\text{o}][\text{d}] - 1000)$ when $\text{ship}[\text{o}][\text{d}] \leq 3000$
- $0.40 * 1000 + 0.20 * 2000 + 0.10 * (\text{ship}[\text{o}][\text{d}] - 3000)$ otherwise

This is an example of a typical piecewise linear function.

Piecewise linear programming in OPL

A **piecewise linear function** is represented by a sequence of discrete segments, each of which is a linear function. A piecewise linear function can be specified in OPL if you know:

- The **slope** of each linear segment
- The **breakpoints** at which the slopes change
- The value of the function at a given point

In the transportation example, this information is as follows:

- Slopes 0.4, 0.2, and 0.1
- Breakpoints 1000 and 3000
- The value 0 at point 0

The OPL syntax is covered in the *Optimization Modeling with IBM ILOG OPL* course.

Integer optimization

Learning objective

Understand how to deal with integer decision variables using IP and MIP, and how these techniques differ from LP.

Key terms

- Integer Programming (IP)
- Mixed Integer Programming (MIP)
- Branch and bound
- Binary variables

Problems requiring integers

For some optimization problems the decision variables should take integer values, for example problems involving:

- production of large indivisible items, such as airplanes or cars
It usually does not make sense to use a continuous variable to represent the number of airplanes to produce, because there is no point in manufacturing a partial airplane.
- a particular state, such as on or off
For example, unit commitment where integer variables are used to represent the state of a particular unit (on or off).
- planning of investments
For example, 1 to invest in a warehouse, and 0 to ignore it
- logic between different decisions
For example, that a given tax break is applicable only if a certain investment is made

Types of integer decisions

Many types of decisions can be modeled using integer variables. Some of these are:

- yes/no decisions (1 for yes, 0 for no):
For example, $x = 1$ indicates that new manufacturing equipment should be installed, and $x = 0$ means that it should not.
- logical decisions (1 for true, 0 for false):
For example, $y_1 = 1$ indicates that the production constraints for machine 1 should be used, and $y_1 = 0$ indicates that the constraints for machine 1 should be ignored.
- State or mode decisions:
For example, $z_1 = 1$ if the machine operates in mode 1, $z_2 = 1$ if the machine operates in mode 2, $z_3 = 1$ if the machine operates in mode 3

The same integer is often used to express both yes/no decisions and logic. For example, $y_1 = 1$ could in this case also be used to indicate that machine 1 is installed, and 0 otherwise.

Types of integer variables

In general, integer variables can take any integer value (0, 1, 2, 3, ...)

Integers that should only take the values of 1 or 0 are known as **0-1** or **binary** variables.

0-1 variables are also often referred to as **Boolean** variables because the Boolean values of true and false are analogous to 1 and 0.

To ensure that an integer variable can only take the values 0 and 1, one can give it an upper bound of 1 or declare it to be a Boolean if the software you use allows such a type for a variable.

In OPL, decision variables are assumed to be nonnegative unless otherwise specified and the lower bound of 0 does not need to be declared explicitly.

Indicator variables

Indicator variables are binary variables used to indicate whether a certain set of conditions is valid (1) or not (0).

For example, consider a production problem where you want to distinguish between two states: 1) production above a minimum threshold, and 2) no production.

Define an indicator variable y to take a value of 1 if production is above the minimum threshold (**minProd**), and 0 if there is no production:

production \geq **minProd** * y

production \leq **maxProd** * y

Here, **maxProd** is an upper bound on the production quantity.

Thus, if $y = 1$, the minimum and maximum production bounds hold:

minProd \leq **production** \leq **maxProd**

And if $y = 0$, the production is set to zero:

$0 \leq$ **production** ≤ 0

Logical constraints

Many problems involve logical constraints that represent logical relationships between decision variables, such as:

- AND: Include ingredient A and ingredient B in the mix.
- OR: Invest in either warehouse 1 or warehouse 2.
- NOT: If you invest in warehouse 1, do not invest in warehouse 2.
- IMPLY: If machine 1 is operating, it implies that it must be operating in exactly one of its operating modes.

Integer variables are ideal for formulating these types of constraints.

For example, consider an investment decision involving a production plant and two warehouses.

If the production plant is invested in, then either warehouse 1 or warehouse 2 may be invested in (not both).

If the production plant is not invested in, then neither of the two warehouses may be invested in.

Let y_{Plant} be 1 if you decide to invest in the production plant, and 0 otherwise. Similar for $y_{\text{Warehouse1}}$ and $y_{\text{Warehouse2}}$. Then this example can be modeled as follows:

$y_{\text{Warehouse1}} + y_{\text{Warehouse2}} \leq y_{\text{Plant}}$

IP versus MIP

When all the decision variables in a linear model should take integer values, the model is an Integer Program (IP).

When some of the decision variables may also take continuous values, the model is a Mixed Integer Program (MIP).

MIPs are very common in, for example, some supply chain applications where investment decisions may be represented by integers and production quantities are represented by continuous variables.

Solving IPs and MIPs

IPs and MIPs are generally much more difficult to solve than LPs.

The solution complexity increases with the number of possible combinations of the integer variables, and such problems are often referred to as being “combinatorial”.

In the worst case, the solution complexity increases exponentially with the number of integer decision variables.

Many advanced algorithms exist that can solve complex IPs and MIPs in reasonable time.

Practice



An integer programming example

In the telephone production problem where the optimal solution found in the lesson on LP had integer values, it is possible that the solution becomes non-integer under certain circumstances, for example:

- Change the availability of the assembly machine to 400.73 hours
- Change the painting machine availability to 490.67 hours
- Change the profit for a desk phone to 12.4
- Change the profit for a cell phone to 20.2

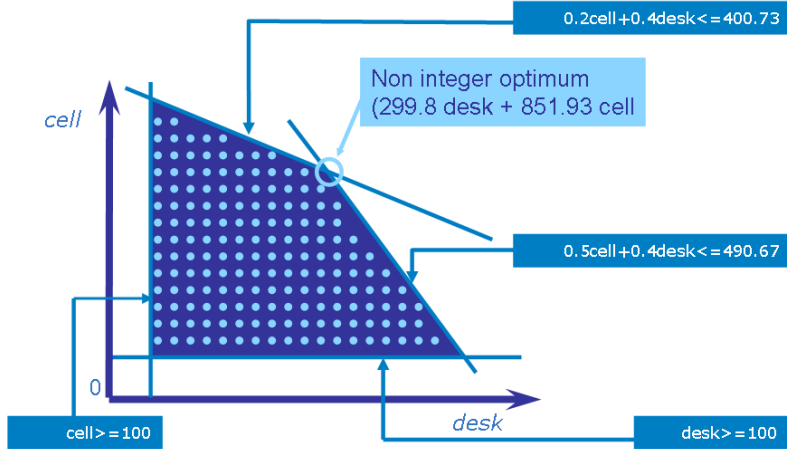
The fractional values for profit are quite realistic. Even though the fractional times for availability are not entirely realistic, these are used here to illustrate how fractional solutions may occur.

If one were to solve this problem using OPL, it would now return the following non-integer solution:

desk = 299.8
cell = 851.93

In reality, you cannot produce 299.8 desk phones and 851.93 cell phones. One way to deal with this is to use integer decision variables. The following graphic (not according to scale) shows the new feasible region where the dots indicate the feasible solutions (solutions where the variables take only integer values).

The telephone problem: a set of integer solutions



Rounding a fractional solution

An idea that often comes up to deal with fractional solutions is rounding.

However, because the optimal solution is always on the edge of the feasible region, rounding can lead to an infeasible solution.

In the case of the telephone problem, rounding would produce infeasible results for both types of phones.

When large quantities of items are produced, for example thousands of phones, rounding may be still be a good approach to avoid integer variables.

The branch and bound method

The **branch and bound** method, implemented in CPLEX, provides an efficient way to solve IP and MIP problems.

This method begins by relaxing the integer requirement and treating the problem as a normal LP problem. If all the variables take integer values, the problem is solved.

If not, it begins a tree search, as illustrated in the following example.

Branch and bound

Consider the following integer linear program:

$$\text{Maximize } x + y + 2z$$

$$\text{Subject to } 7x + 2y + 3z \leq 36 \quad (\text{IP}_0)$$

$$5x + 4y + 7z \leq 42$$

$$2x + 3y + 5z \leq 28$$

$$x, y, z \geq 0, \text{ integer}$$

Branch and bound

1. Solve as an LP $\begin{pmatrix} x = 1\frac{3}{11} \\ y = 0 \\ z = 5\frac{1}{11} \end{pmatrix}$ $obj = 11\frac{5}{11}$ IP_0 Result is fractional

Best IP value $-\infty$

Best IP solution

Branch and bound

1. Solve as an LP $\begin{pmatrix} x = 1\frac{3}{11} \\ y = 0 \\ z = 5\frac{1}{11} \end{pmatrix}$ $obj = 11\frac{5}{11}$ IP_0 Result is fractional

$\begin{pmatrix} 1 \\ 0 \\ 5.2 \end{pmatrix}$ $obj = 11.4$

2. Search 2 possible integer values for x (fractional)

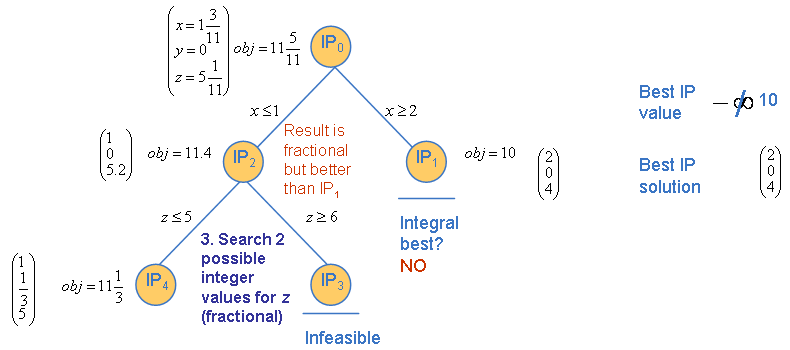
IP_1 $obj = 10$ $\begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$

Integral best?

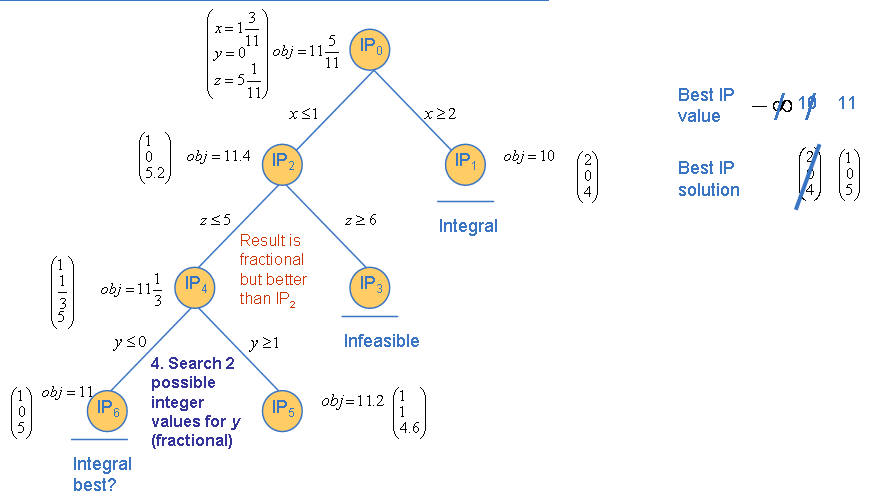
Best IP value $-\infty$ 10

Best IP solution $\begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$

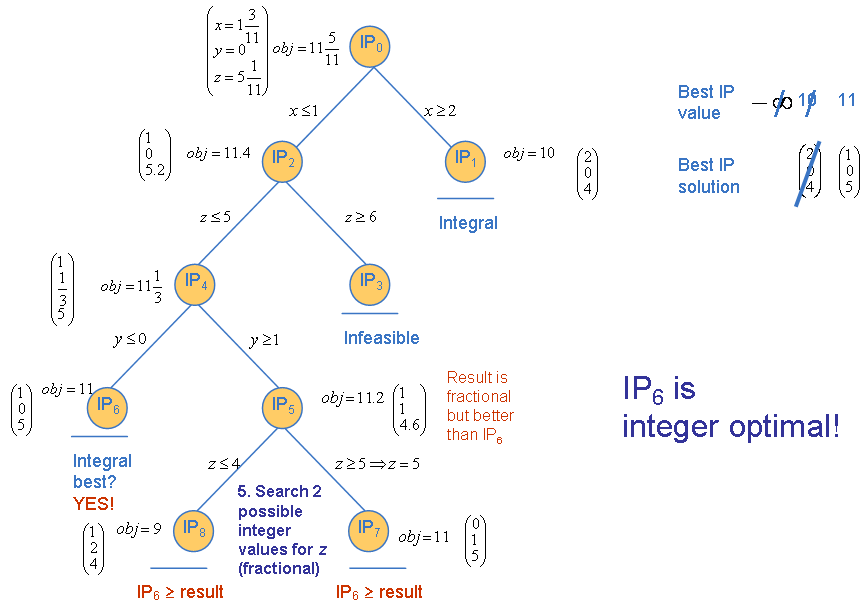
Branch and bound



Branch and bound



Branch and bound



Practice



Use Booleans to model yes/no decisions

Consider again the telephone production problem, but ignore the lower bounds of 100 on production for simplicity:

$$\max 12 * \text{desk} + 20 * \text{cell}$$

subject to

$$\text{Assembly machine 1: } 0.2 * \text{desk} + 0.4 * \text{cell} \leq 400$$

$$\text{Painting machine: } 0.5 * \text{desk} + 0.4 * \text{cell} \leq 490$$

$$\text{desk} \geq 0, \text{ cell} \geq 0$$

The company is considering replacing the assembly machine with a newer machine that requires less time for cell phones (18 minutes), but more time for desk phones (15 minutes). This machine is available for 430 hours, because it requires less downtime.

Write a model on paper that uses a Boolean variable to help the company choose between the two machines. Try to do this yourself before looking at the solution.

The steps to formulate the mixed-integer model are:

1. Define four new variables (**desk1**, **desk2**, **cell11**, and **cell12**) to indicate the production on assembly machines 1 and 2, respectively.
2. Add two constraints to define the total production of **desk** and **cell** to equal the sum of production from the two assembly machines.
3. Rewrite the constraint for assembly machine 1 to use the new variables for that machine (**desk1** and **cell11**).
4. Add a similar constraint for the production on assembly machine 2.
5. Define a Boolean variable, **y**, to take a value of 1 if assembly machine 1 is chosen, and 0 if assembly machine 2 is chosen.
6. Use the **y** variable to set the production to zero for the machine that is not chosen.

Solution:

$$\max 12 * \text{desk} + 20 * \text{cell}$$

subject to

$$\text{desk} = \text{desk1} + \text{desk2}$$

$$\text{cell} = \text{cell11} + \text{cell12}$$

$$\text{Assembly machine 1: } 0.2 * \text{desk1} + 0.4 * \text{cell11} \leq 400$$

$$\text{Assembly machine 2: } 0.25 * \text{desk2} + 0.3 * \text{cell12} \leq 430$$

$$\text{Painting machine: } 0.5 * \text{desk} + 0.4 * \text{cell} \leq 490$$

$$\text{desk1}, \text{cell11} \leq U * y$$

$$\text{desk2}, \text{cell12} \leq U * (1 - y)$$

$$\text{desk}, \text{desk1}, \text{desk2}, \text{cell}, \text{cell11}, \text{cell12} \geq 0$$

Test whether your model will behave correctly by setting **y** to 1 or 0.

U is an upper bound on the production quantities. You should try and use as tight upper bounds as possible, without adding additional constraints on the production. For the tightest bounds, you would calculate upper bounds on **desk1**, **desk2**, **cell11**, and **cell12** individually, as opposed to using a global upper bound as done here.

The following constraints would also have the effect of setting the production to zero for the machine that is not chosen:

$$\text{Assembly machine 1: } 0.2 * \text{desk1} + 0.4 * \text{cell11} \leq 400 * y$$

$$\text{Assembly machine 2: } 0.25 * \text{desk2} + 0.3 * \text{cell12} \leq 430 * (1 - y)$$

However, these constraints are generally not as efficient as setting the production quantities to zero individually.

For this simple example, one could've formulated two separate LPs (one for using machine 1 and one for using machine 2) and solved them separately to compare the solutions. However, this would be impractical if several yes/no decisions must be made in combination. Keep in mind that MIP algorithms, such as the branch-and-bound algorithm, use rules to avoid solving all possible LPs resulting from a MIP formulation, and are generally much more efficient than attempting to enumerate all possible outcomes.

Write constraints that use Boolean variables for logical decisions

Consider the case where the phone company has to choose between 3 possible candidates for the assembly machine.

Write a logical constraint that forces exactly one possible candidate to be chosen.

Solution:

Let y_1 , y_2 , and y_3 represent choosing machines 1, 2, and 3, respectively. If $y_1 = 1$, then machine 1 is chosen, and so forth.

To force exactly one machine to be chosen, the other two Booleans must take values of zero when the chosen Boolean takes a value of 1.

The following constraint enforces this:

$$y_1 + y_2 + y_3 = 1$$

If $y_1 = 1$, the only possible value for y_2 and y_3 that would satisfy this constraint is 0.

Quadratic programming

Learning objective

Learn what a QP is and how it can be used to calculate stock portfolio risk

Key terms

- covariance
- Quadratic Program (QP)
- Quadratically Constrained Program (QCP)

What is a quadratic function?

Mathematically, a function is quadratic if:

- The variables are only first or second degree (that is, one variable may be multiplied by another variable, and any of the variables may be squared) and
- The coefficients of the variables are constant numeric values (that is, integers or real numbers).

A quadratic function is also known as a second degree polynomial function.

Geometrically, a quadratic function is a curve or curved surface.

For example, a quadratic function in two dimensions is a curved line, such as a parabola, hyperbola, ellipse, or circle.

What is a Quadratic Program (QP)?

QPs have quadratic objectives and linear constraints.

A model that has quadratic functions in the constraints is a Quadratically Constrained Program (QCP). The objective function of a QCP problem may be quadratic or linear.

A simple formulation of a QP is:

$$\text{minimize } 1/2 \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{c}^T \mathbf{x}$$

$$\text{subject to } \mathbf{A} \mathbf{x} \geq \mathbf{b}$$

$$l \leq \mathbf{x} \leq \mathbf{ub}$$

Here, \mathbf{Q} is the matrix of objective function coefficients of the quadratic terms.

For example, q_{11} is the coefficient for a quadratic term in x_1 ($q_{11}x_1^2$) and q_{12} is the coefficient for a bilinear term of x_1 and x_2 ($q_{12}x_1x_2$).

CPLEX can solve convex QP and QCP problems.

Many optimization problems, such as portfolio management or chemical process modeling often require the use of Quadratic Programming (QP).

Portfolio management

In order to mitigate risk while ensuring a reasonable level of return, investors purchase a variety of securities and combine these into an investment portfolio.

Each security has an expected return and an associated level of risk (or variance).

Securities sometimes **covary**, that is, they change together with some classes of securities, and in the opposite direction of other classes of securities.

To optimize a portfolio in terms of risk and return, an investor will evaluate the following:

- Sum of expected returns of the securities
- Total variances of the securities
- Covariances of the securities

An example of positive covariance is when shares in technology companies follow similar patterns of increases and decreases in value.

On the other hand, as the price of oil rises, shares in oil companies may increase in value, but plastics manufacturers, who depend on petroleum as a major primary resource, may see their shares decline in value as their costs go up and vice versa. This is negative covariance.

A portfolio that contains a large number of positively covariant securities is more risky (and potentially more rewarding) than one that contains a mix of positively and negatively covariant securities.

The role of portfolio optimization

Portfolio optimization is used to select securities to maximize the rate of return, while managing the volatility of the portfolio and remaining within the investment budget.

As the securities covary with one another, selecting the right mix of securities can change or even reduce the volatility of the portfolio with the same expected return.

At a given expected rate of return, there is one portfolio which has the lowest risk. If you plot each lowest-risk portfolio for each expected rate of return, the result is a convex graph, called the efficient frontier.

The risk-return characteristics of a portfolio change in a nonlinear fashion, and **quadratic expressions** are used to model them.

You can find an example of a portfolio optimization model with the OPL distribution at `<install_dir>/examples/opl/models/Portfolio`.

`<install_dir>` is the directory where OPL is installed.

Summary

Review

Network structures, consisting of nodes and arcs, are often found in practical applications. Such structures can often be optimized using LP, and some common types of network problems are:

- Transportation problems
- Transshipment problems
- Assignment problems
- Shortest path problems
- Critical path analysis problems

CPLEX includes specialized network algorithms.

Some problems have constraints that can only be modeled using nonlinear functions. Nonlinear functions are either convex or nonconvex. A feasible region is convex if the portion of the straight line between any two points in the region also lies in the region. All LPs are convex.

Convex nonlinear functions can often be approximated by a series of linear functions, known as a piecewise linear function.

Piecewise linear programming is also useful to model problems such as price changes due to economies of scale.

Integer variables may be necessary to, for example:

- Deal with discrete, indivisible items.
- Indicate certain states, often linked to other, continuous decision variables (using zero-one, or binary decision variables).
- Determine which of a number of possible decisions should be taken.

Integer Programs (IPs) are linear optimization problems that contain only integer decision variables. Mixed Integer Programs (MIPs) are linear optimization problems that contain a mix of integer and continuous decision variables.

The best known algorithm for solving IPs and MIPs is the branch-and-bound algorithm.

Certain types of problems, for example, stock portfolio risk analysis, can only be solved using quadratic expressions. Quadratic programs can have a quadratic objective function and linear constraints, in which case it is a Quadratic Program (QP). It can also have either quadratic or linear objective function, and quadratic constraints, in which case it is a Quadratically Constrained Problem (QCP). Both require special solution algorithms which are available with CPLEX.

This page is intentionally left blank.

Lesson 4: Modeling Practice

At the end of this lesson you will be able to:

- Explain under which circumstances a fractional solution can be rounded.
- Explain the role of uncertainty in optimization.
- Explain what a multiperiod model is and why this type of model is used.
- Explain how penalties can be used to prioritize objectives.
- List some tips and tricks for optimization modeling in the real world.
- Explain what sparsity is, in why it is important.
- Formulate a simple LP model.

Modeling in the real world

Learning objective

Look at the differences between a model and the real-world system it represents.

Key terms

- uncertainty
- multi-period model
- soft constraint
- penalty

Real-world optimization models may be difficult to solve, or it may be difficult to represent the exact business conditions in a mathematical model. In this topic you'll learn about some techniques that help deal with some of the challenges associated with real-world models.

Rounding fractional solutions

In the telephone example given earlier in this training, you learned that rounding a fractional result could lead to an infeasible solution.

However, when the variable represents large numbers of a single item, such as thousands of telephones, it is usually acceptable to round in order to avoid integer variables.

The telephone production problem could be modeled as an LP, and the fractional solutions could be rounded down.

That way, the increased computational effort associated with an integer solution is avoided without a significant loss to the quality of the solution.

On the other hand, if the model is for a production process of a small number of high-value items, such as the production of airplanes, an error of even one unit can be very costly and it is better to use integer variables.

Uncertainty

Uncertainty often plays an important role in real-world decision making. Sources of uncertainty are, for example:

- Fluctuations in demand
- Price changes
- Exchange rate fluctuations
- Machine breakdowns
- Labor disputes and strikes

Sometimes a reasonable solution to an optimization problem requires that uncertainty be considered to some extent.

For example, an oil company planning multi-billion dollar investments would want to incorporate the uncertainty associated with oil price fluctuations.

Incorporating uncertainty into a model typically results in a significant increase in complexity.

Several sophisticated techniques exist for incorporating uncertainty into optimization models, for example Stochastic Programming and Robust Optimization.

In many cases uncertainty can be dealt with by solving multiple scenarios of the same optimization model, and using a different data instance for each scenario.

Multiperiod models

The example models in this training deal with one time period, or don't take time into consideration.

In many real-life applications, it is important to take periodicity into account: business conditions vary with time and often decisions need to be made on a periodic basis while considering the future.

Multi-period models are used to deal with conditions that change over time.

These models use an index for each relevant variable and data element to represent each time period.

For example, a **production** variable in a single-period model will become **production(t)** in a multi-period model, where the index **t** indicates the time period. In addition, some constraints are usually required to tie each period to the preceding one.

A typical case of multiperiod modeling is a supply chain model where decisions to produce or ship a given product are typically made on a daily or weekly basis.

In addition, future demand and inventory levels need to be taken into account.

For example, the inventory in a current period (**t**) can be expressed as follows in relation to the previous period (**t-1**):

$$\mathbf{inventory(t) = inventory(t-1) + production(t) - customer_shipments(t)}$$

The choice of time units for the period is important when building a multi-period model, because the model should not get unnecessarily large.

For example, when decisions are made on a weekly basis, it is unnecessary to have hourly increments.

Sometimes a model may include several different time units: the near future may be in short increments, while the far future is in longer increments to reflect the fact that the far future is more uncertain.

Using penalties for prioritization

Priorities for business objectives can be addressed by using penalties associated with each of these objectives in the objective function.

For example, if it is impossible to supply all customers, a company may want to prioritize an older customer even though that customer doesn't pay as much as a newer one.

In this case, you can add an additional penalty for not satisfying a favorite customer.

The more such penalties present in a model, the more difficult it will be to find a balance between the different goals and these should therefore be used only when absolutely necessary and the trade-offs are well understood.

The importance of sparsity

Learning objective

Learn what sparsity means in an OR context, and how to ensure sparse models and data.

Key terms

- sparsity
- sparse data
- sparse models

Definition of sparsity

In an OR context, **sparsity** refers to the characteristic where a matrix representation of an optimization model has many elements that have 0 (zero) as a value. In the following diagram, the matrix on the left would be considered sparse, while the matrix on the right would be considered non-sparse or “dense”.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix} \quad \text{vs.} \quad \begin{bmatrix} 0 & 1 & 2 & 1 \\ 2 & 1 & 0 & 3 \\ 1 & 1 & 3 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

Very large LP models are often sparse, for example transportation problems. In such cases, one can exploit the sparse structure to dramatically reduce the time and memory required for setting up and solving the optimization model. It is therefore important to ensure from the start that both the data and the model are treated in a way that exploits the sparsity.

Model sparsity

Consider a problem where three products can be shipped between three suppliers and two customers. The decision variable $\mathbf{x}(\mathbf{p}, \mathbf{s}, \mathbf{c})$ represents the amount of product \mathbf{p} to be shipped between each supplier, \mathbf{s} , and each customer, \mathbf{c} .

A typical constraint for such a problem could be that the total shipments to a customer should exceed that customer's demand, and in this case this would lead to a total of 6 constraints as follows:

$$\begin{aligned} x_{111} + x_{121} + x_{131} &\geq demand_{11} \\ x_{211} + x_{221} + x_{231} &\geq demand_{21} \\ x_{311} + x_{321} + x_{331} &\geq demand_{31} \\ x_{112} + x_{122} + x_{132} &\geq demand_{12} \\ x_{212} + x_{222} + x_{232} &\geq demand_{22} \\ x_{312} + x_{322} + x_{332} &\geq demand_{32} \end{aligned}$$

When coding these constraints using software, one would not write out each constraint, but instead use a compact representation. A compact, yet naïve, representation of these constraints is as follows:

$$sum(s, x_{p,s,c}) \geq demand_{p,c} \quad \text{for all } p, c$$

Now imagine that customer 1 is only interested in products 1 and 2, and customer 2 is only interested in product 3. Suddenly the data has become more sparse. The redundant constraints should be removed so that the constraint set is reduced to:

$$\begin{aligned}
 x_{111} + x_{121} + x_{131} &\geq demand_{11} \\
 x_{211} + x_{221} + x_{231} &\geq demand_{21} \\
 x_{312} + x_{322} + x_{332} &\geq demand_{32}
 \end{aligned}$$

However, if you were to use the compact representation given previously, you would end up with the original set of 6 constraints instead of 3. This is because the constraints are defined to be valid for all products, **p**, and all customers, **c**, leading to 6 possible combinations.

To properly express the sparsity of the model, you should instead define a new object to represent all possible combinations of products and customers, and define the constraints for all such combinations as follows:

$$sum(s, x_{p,s,c}) \geq demand_{p,c} \quad \text{for all validCombinations}(p, c)$$

By defining the constraints only over the valid combinations, you ensure that the redundant constraints are excluded from the model, thus freeing up memory and leading to quicker solutions. OPL allows users to do this through the use of data structures called *tuples*. You will learn more about tuples and their effect on sparsity in the OPL course.

Data sparsity

Consider again this same set of constraints, but from the perspective of data representation. The naïve approach would be to declare the demand data over all products and customers, **demand(p)(c)**, and import an associated data set:

	Products	Customers	Demand
	P1	C1	10
	P2	C1	20
	P3	C1	0
	P1	C2	0
	P2	C2	0
	P3	C2	15

It is clear that the data for the invalid combinations not only unnecessarily occupies database space, but will also add unnecessary time and memory requirements to data import and model setup. Instead, the demand should be declared over only the valid combinations **demand(validCombinations(p, c))** with the more compact associated data set:

	Products	Customers	Demand
	P1	C1	10
	P2	C1	20
	P3	C2	15

A packing model for “what-if” analysis

Learning objective

Use optimization language to model a problem stated in business terms, and use that model for “what-if” analysis.

Key terms

- production rate
- time constraint
- “what-if” analysis

In this topic, you'll practice:

- writing an optimization model on paper
- writing the same model using OPL, with guidance from your instructor
- using the model to perform “what-if” analysis

Most of the model is already completed in OPL and you may be able to copy the syntax of the completed parts to construct the remainder of the model.

While this training includes some brief explanations of the OPL syntax, this is only covered thoroughly in the main course – **don't hesitate to ask your instructor for help in understanding**

the OPL syntax.

A packing model: Problem description

Beverage Ltd. manufactures two beverage products, namely **sportsDrink** and **classicDrink**. This practice focuses on the packing department of their manufacturing facility, where there are two packing machines, namely **packer1** and **packer2**, that are used to pack the bottled beverages into cases.

Each of the products can be packed on either machine, but the machines are not identical: **packer1** is older and slower, with the advantage of a lower hourly cost associated with using this machine.

Beverage Ltd.'s major objective is to minimize manufacturing cost while satisfying demand. They have hired you to write a model that will determine how much of each product to produce on each machine in order to achieve their goal during a typical week of production. They may also want to use this model for “what-if” analysis, but you will get back to this later.

Requirements:

- Manufacture products during the time available
- Meet customer demand
- Consider the different production rates associated with each product on each machine
- Minimize manufacturing cost

The complete data set is given next in the workbook.

Details of the problem

The **Products** are:

- sportsDrink
- classicDrink

The **Machines** used to pack the Products are:

- packer1
- packer2

Each **Machine** requires some downtime during the week for maintenance, leaving the following **time available** for packing during a typical week:

- packer1: 104 hours
- packer2: 112 hours

The **production rates** of each **Product** on each **Machine** are as follows:

- packer1:
 - sportsDrink: 20 cases/hr
 - classicDrink: 10 cases/hr
- packer2:
 - sportsDrink: 40 cases/hr
 - classicDrink: 30 cases/hr

The **cost** to pack each product is associated with the time each machine is in use, and are as follows:

- 50 \$/hour for product packed on packer1
- 70 \$/hour for product packed on packer2

Finally, the expected **demand** for each product during the week is as follows:

- sportsDrink: 3200 cases
- classicDrink: 2000 cases

Practice



Formulate the model on paper before starting the OPL part of the workshop.

1. Discuss with your colleagues and your instructor how the business requirements can be transformed into an optimization model. You need to define the following on paper in mathematical form:
 - Data elements
 - Decision variables
 - Objective function
 - Constraints
2. Once you've completed the model on paper, go to the **Beverage packing** workshop and perform step 1, **Formulate the model**

Now that you've completed the model, you can start using OPL to perform “what-if” analysis to deal with the following scenarios:

- Scenario 1: An unanticipated heat-wave when demand is expected to jump significantly
- Scenario 2: An investment decision for a new packing machine

This is an advanced exercise – only attempt it if you are comfortable with what you've done thus far. Refer to the problem description listed next in the workbook.

Go to the **Beverage Packing** workshop and perform step 2: **Perform “what-if” analysis.**

- Scenario 1: How to react in the event of an unanticipated heat-wave when demand is expected to jump to 5000 for **sportsDrink** and 3500 for **classicDrink**. Beverage Ltd. would like to prioritize **classicDrink** because it is their “flagship” product.
- Scenario 2: Beverage Ltd. is considering replacing the current packers with a new **superPacker** that packs much faster and doesn't require any maintenance, but has a higher hourly cost associated with it. They want to know whether to go ahead with this replacement, based on a regular week of demand. The data associated with the **superPacker** is as follows:
 - Production rate for **sportsDrink**: 50 cases/hr
 - Production rate for **classicDrink**: 60 cases/hr
 - Cost: 110 \$/hour
 - Availability: 120 hours

Tips for better models

Learning objective

Learn some tips to construct well-design models.

Key terms

- Simplicity
- Granularity
- Scaling

To conclude this course, here are a few tips that may help you construct better models.

Keep it simple

Keep your models as simple and streamlined as possible.

Of course, for very complex situations, the simplest may still be very complex – all the more reason not to complicate a model unnecessarily.

Granularity

Before you start counting lots of very small items in very large numbers, think about what you really need to know. Are you concerned directly with how many individual tennis-balls you need to make, or can you count packages of tennis-balls. Or, perhaps, you only need to work with containers full of tennis-ball packages? Work at the smallest level of granularity that you really need.

Scaling

The scale of numbers used in an LP problem can affect computation time.

As a general guideline, try to avoid extremely large numbers (for example, millions) and extremely small real numbers whenever possible.

Ranges from zero to the thousands work well, but be careful of multiplying thousands by thousands – the results get very large very fast.

You can keep the scale down by dividing your data by 10, 100, and so forth, as appropriate.

Continuous decision variables versus integers

In many cases, it is preferable to use continuous decision variables, even if the problem deals with countable discrete items. This is because linear programming models are usually easier to solve than ones that contain integer decision variables.

Whenever a continuous decision variable will give you a valid, meaningful result, it is better to use than an integer decision variable.

Common sense

Above all, building a model and interpreting the results requires common sense. A model might produce a mathematically optimal result, but that result might not be implementable. For example, one way to successfully relax an infeasible model could be to reduce customer demand. However, a company may place high priority on meeting customer demand and not want such a constraint relaxed.

Constructing and interpreting a model depends on a combination of mathematical programming skills and knowledge of the industry and the situation you are modeling. If you apply the right blend of these qualities, you will create useful models that will help your organization attain its goals.

Summary

Review

There are many different ways to tune a model so that it follows good practice and runs efficiently. Some of these include:

- If you can easily use a result rounded from a real number to a problem involving discrete items, a continuous model will run faster than a MIP model. Use MIP when a rounded result could be misleading.
- Take advantage of sparse models and data.
- Learn to factor in uncertainty, periodicity and soft constraints so that your models reflect the real world.
- Use only the granularity you really need in a model.
- Try to avoid extremely large numbers or extremely small fractional values. Use scaling to reduce computation time.

Above all, use your familiarity with the business situation, and common sense.

Conclusion

In this training, you learned that Mathematical Programming (MP) is a powerful technique used to model and solve optimization problems. This technique was developed by researchers in the area of Operations Research (OR).

You also had an introduction to IBM® ILOG OPL Studio and learned about the two solution engines available in OPL, namely:

- CPLEX – IBM's MP engine
- CP Optimizer – IBM's new constraint-programming optimization engine built on the legacy of IBM ILOG CP.

You have explored aspects of MP that are important for helping you to collaborate with OR experts or to start learning more about constructing optimization models in OPL. Among these are:

- Definition of basic elements of a model:
 - Data
 - Decision variables
 - Objective function
 - Constraints
- Types of solutions:
 - Feasible
 - Optimal
 - Infeasible
 - Unbounded
- MP techniques for optimization:
 - Linear Programming (LP)
 - Integer Programming (IP)
 - Mixed Integer Programming (MIP)
 - Quadratic Programming (QP)
- Algorithms used for solving continuous LP problems:
 - simplex
 - dual simplex
 - barrier
- Other topics:
 - Network models
 - Piecewise Linear Programming
 - Nonlinearity and convexity

You have also learnt some tips for good modeling practice, and reviewed some of the history and background of Operations Research.

The next step is to learn more about using OPL. The training course, *Optimization modeling with IBM ILOG OPL*, is a 3-day course providing hands-on experience constructing OPL models and learning about its features. It will also introduce OPL's connection with IBM ILOG Optimization Decision Manager (ODM) for application building and real-time what-if analysis.

For more information:

<http://www.ilog.com>

Technical support pages:

Open service requests:

http://www.ibm.com/support/electronic/uprtransition.wss?category=2locale=en_us.

This is a tool to help clients find the right place to open any problem, hardware or software, in any country where IBM does business. This is the starting place when it is not evident where to go to open a service request.

Service Request (SR): <http://www.ibm.com/software/support/probsub.html>

This page offers Passport Advantage clients for distributed platforms online problem management to open, edit and track open and closed PMRs by customer number.

You can find information about assistance for SR at
<http://www.ibm.com/software/support/help-contactus.html>