Overview of Xpress

- Optimization algorithms
  - solve different classes of problems
  - built for speed, robustness and scalability
- Modeling interfaces
  - Mosel
    - formulate model and develop optimization methods using Mosel language / environment
  - BCL
    - build up model in your application code using object-oriented model builder library
- Application development
  - Insight
    - deploy multi-user optimization applications
Xpress users

- Supply chain optimization
- Portfolio generation + maintenance
- Personnel scheduling
- Production scheduling
- Process industries (blending)
- Vehicle routing

- Marketing optimization
- Generalized assignment (e.g. facility location)
- Auctions
- Airline operations
- Electricity generation, supply and pricing

1 Application design
2 Modeling platforms

Why use modeling software?

- Developing a working model is the difficult bit
- Important to have software that helps
  - speed to market
  - verify correctness
  - maintenance & modification
  - algorithmic considerations
  - execution speed

Modeling platforms
Xpress modeling interfaces

- Mosel
  - formulate model and develop optimization methods using Mosel language / environment
- BCL
  - build up model in your application code using object-oriented model builder library
- Optimizer
  - read in matrix files
  - input entire matrix from program arrays

Mosel

- A modeling and solving environment
  - integration of modeling and solving
  - programming facilities
  - open, modular architecture
- Interfaces to external data sources (e.g. ODBC, host application) provided
- Language is concise, user friendly, high level
- Best choice for rapid development and deployment

Xpress-BCL

- Model consists of BCL functions within application source code (C, C++, Java, C# or VB)
- Develop with standard C/C++/Java/C#/VB tools
- Provide your own data interfacing
- Lower level, object oriented approach
- Enjoy benefits of structured modeling within your application source code

Xpress-Optimizer

- Model is set of arrays within application source code (C, Java, C#, or VB)
- May also input problems from a matrix file
- Develop with standard C/C#/Java/VB tools
- Provide your own data interfacing
- Very low level, no problem structure
- Most efficient but lose easy model development and maintenance
3 Xpress-Mosel

- A high-level modeling language combined with standard functionality of programming languages
  - implementation of models and solution algorithms in a single environment
- Open, modular architecture
  - extensions to the language without any need for modifications to the core system
- Compiled language
  - platform-independent compiled models for distribution to protect intellectual property

...and also

- Mosel modules
  - solvers: mmxprs, mmquad, mmxlnp, mmnl, kalis
  - data handling: mmetc, mmmodbc, mmoci
  - model handling, utilities: mmjobs, mmsystem
  - graphics: mmive, mmxad
- IVE: visual development environment (Windows)
- Library interfaces for embedding models into applications (C, Java, C#, VB)
- Tools: debugger, profiler, model conversion, preprocessor

Example: Portfolio optimization

- An investor wishes to invest a certain amount of money into a selection of shares.
- Constraints:
  1. Invest at most 30% of the capital into any share.
  2. Invest at least half of the capital in North-American shares.
  3. Invest at most a third in high-risk shares.
- Objective: obtain the highest expected return on investment

Portfolio optimization: Mathematical model

\[
\begin{align*}
\text{maximize} & \quad \sum_{s \in \text{SHARES}} \text{RET}_s \cdot \text{frac}_s \\
\sum_{s \in \text{RISK}} \text{frac}_s & \leq \frac{1}{3} \\
\sum_{s \in \text{NA}} \text{frac}_s & \geq 0.5 \\
\sum_{s \in \text{SHARES}} \text{frac}_s & = 1 \\
\forall s \in \text{SHARES} : 0 \leq \text{frac}_s \leq 0.3
\end{align*}
\]
Portfolio optimization: Mosel model

model "Portfolio optimization with LP"
uses "mxmlprs" ! Use Xpress-Optimizer

declarations
SHARES = 1..10 ! Set of shares
RISK = {2,3,4,9,10} ! Set of high-risk shares
NA = {1,2,3,4} ! Set of shares issued in N.-America
RET: array(SHARES) of real ! Estimated return in investment
frac: array(SHARES) of mpvar ! Fraction of capital used per share
end-declarations

RET: [5,17,26,12,8,9,7,6,31,21]
sum(s in RISK) frac(s) <= 1/3 ! Limit percentage of high-risk
sum(s in NA) frac(s) >= 0.5 ! Min. amount of North-American
forall(s in SHARES) frac(s) = 1 ! Spend all the capital
forall(s in SHARES) frac(s) <= 0.3 ! Bound on investment per share
maximize(sum(s in SHARES) RET(s)*frac(s)) ! Solve the problem
writeln("Total return: ", getobjval) ! Solution printing
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, ")")
end-model

Portfolio optimization: Logical Conditions (MIP)

1. Binary variables

declarations
frac: array(SHARES) of mpvar ! Fraction of capital used
buy: array(SHARES) of mpvar ! 1 iff asset is in portfolio
end-declarations

forall(s in SHARES) do
frac(s) <= buy(s) ! Linking the variables
end-do

sum(s in SHARES) buy(s) <= MAXNUM ! Limit total number of assets

2. Semi-continuous variables

forall(s in SHARES) do
frac(s) <= MAXVAL ! Upper bound on investment
frac(s) is_semcont MINVAL ! Lower bound on investment
end-do

Portfolio optimization: Extension: Solving iterations

• We wish to
  1. run the model with different limits on the portion of high-risk shares,
  2. represent the results as a graph, plotting the resulting total return against the deviation as a measure of risk.

• Algorithm:
  - for every parameter value
    • re-define the constraint limiting the percentage of high-risk values,
    • solve the resulting problem,
    • if the problem is feasible: store the solution values.

declarations
SOLRET: array(range) of real ! Solution values (total return)
SOLDEV: array(range) of real ! Solution values (average deviation)
end-declarations

! Solve the problem for different limits on high-risk shares
cnt:=0
forall(r in 0..20) do
Risk:= sum(s in RISK) frac(s) <= r/20 ! Redefine high-risk limit
maximize(Return) ! Solve the problem
if (getprobstat = XPRS_OPT) then ! Save the optimal solution value
cnt:=1
SOLRET(cnt):= getobjval
SOLDEV(cnt):= getsol(sum(s in SHARES) DEV(s)*frac(s))
else
end
writeln("No solution for high-risk values <= *, 100*r/20, "%")
end-if
end-do

Portfolio optimization: Extension: Graph drawing

! Drawing a graph to represent results (plot1) and data (plot2, plot3)
declarations
plot1, plot2, plot3: integer
deadclarations
plot1 := IVEaddplot("Solution values", IVE_BLACK)
plot2 := IVEaddplot("Low risk", IVE_YELLOW)
plot3 := IVEaddplot("High risk", IVE_RED)
forall (r in 1..ct) IVEdrawpoint(plot1, SOLRET(r), SOLDEV(r))
forall (r in 2..ct) IVEdrawline(plot1, SOLRET(r-1), SOLDEV(r-1), SOLRET(r), SOLDEV(r))
forall (s in SHARES - RISK) do
IVEdrawpoint(plot2, RET(s), DEV(s))
IVEdrawlabel(plot2, RET(s)+3.4, 1.3*(DEV(s)-1), s)
end-do
forall (s in RISK) do
IVEdrawpoint(plot3, RET(s), DEV(s))
IVEdrawlabel(plot3, RET(s)-2.5, DEV(s)-2, s)
end-do

Portfolio optimization: Extension: Quadratic Programming

- Minimize the risk whilst obtaining a certain target yield using estimates of the variance/covariance matrix of estimated returns on the securities (Markowitz model).
  1. Minimize the variance subject to getting some specified minimum target yield. \( \Rightarrow QP \)
  2. Which is the least variance investment strategy if choosing at most four different securities (again subject to getting some specified minimum target yield)? \( \Rightarrow MIQP \)
  3. Which is the highest return that can be achieved when limiting the total variance to 0.55? \( \Rightarrow QCQP \)

model "Portfolio optimization with nonlinear constraints"
uses "mmxnlp" ! Use Xpress-Nonlinear
... 
declarations
frac: array(SHARES) of mpvar ! Fraction of capital used per share
deadclarations
! Objective: total return
Return:= sum(s in SHARES) RET(s)*frac(s)
! Limit variance
sum(s,t in SHARES) VAR(s,t)*frac(s)*frac(t) <= MAXVAR
...
! Solve the problem
maximize(Return)
end-model
Portfolio optimization: Extension: Solution enumeration

maximize(XPRS_ENUM, Return) ! Enable solution enumerator + solve

! Print out all solutions saved by the enumerator
forall(i in 1..getparam("XPRS_enumsols")) do
  selectsol(i) ! Select a solution from the pool
  writeln("Solution ", i, ". Total return: ", getobjval)
  forall(s in SHARES | getsol(frac(s))>0)
    writeln(s, ": ", getsol(frac(s))*100, ", (", getsol(buy(s)), ")")
end-do

Standard MIP search: Solution enumerator:

4 Advanced data handling features

4.1 I/O drivers

Data handling

- Physical files:
  - text files (Mosel format, new: binary format, diskdata; free format, new: XML,
  - spreadsheets (new: generic spreadsheets, CSV), databases (ODBC or specific drivers)
- In memory:
  - memory block/address
  - streams; pipes; callbacks
- ⇒ Notion of I/O driver
  - change of the data source = change of the I/O driver

Example: Portfolio optimizationExcel data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>Data ranges used by &quot;foloexcel.xls&quot;:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SHARES</td>
<td>SET</td>
<td>RISK</td>
<td>NA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>hardware</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>software</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>telecomm</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>oil</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>energy</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>gas</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>food</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>service</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

parameters
DATAFILE = "msm.shet.excel:folio.xls" ! Extended file name
DBDATA = "foliodata" ! Problem data
DBSOL = "grow;folioresult" ! Solution data
end-parameters

initializations from DATAFILE
[SET,RISK,NA] as DBDATA
end-initializations
...

declarations
Solfrac: array(SHARES) of real ! Solution values
end-declarations
forall (s in SHARES) Solfrac(s) := getsol(frac(s)) * 100

initializations to DATAFILE
Solfrac as DBSOL
end-initializations

Example: Portfolio optimization
Generic spreadsheet access

parameters
DATFILE = "mmsheet.xls:folio.xls" ! Extended file name
DBDATA = "foliodata" ! Problem data
DBSOL = "grow;folioresult" ! Solution data
end-parameters

initializations from DATAFILE
[RET,RISK,NA] as DBDATA
end-initializations

... declarations
Solfrac: array (SHARES) of real ! Solution values
end-declarations

forall (s in SHARES) Solfrac(s) := getsol(frac(s)) * 100

initializations to DATAFILE
Solfrac as DBSOL
end-initializations

Example: Portfolio optimization
In-memory data exchange with Java

parameters
DATFILE = "jraw:" ! Extended file name
DBDATA = "folio(ind,ret,ris,na)" ! Problem data
DBSOL = "folioresult(ind,val)" ! Solution data
end-parameters

initializations from DATAFILE
[RET,RISK,NA] as DBDATA
end-initializations

... declarations
Solfrac: array (SHARES) of real ! Solution values
end-declarations

forall (s in SHARES) Solfrac(s) := getsol(frac(s)) * 100

initializations to DATAFILE
Solfrac as DBSOL
end-initializations

public static class MyData {
    public String ind; // index name
    public double ret; // data value
    public integer risk, na; // data values
}

public static class MySolArray {
    public String ind; // index name
    public double val; // solution value
}

MyData foliodata[] = ...;
MySolArray[] solfrac = new MySolArray[maxnum]; // Associate the Java objects with names in Mosel
mosel.bind("folio", foliodata);
mosel.bind("folioresult", solfrac);
// Pass model parameters through execution parameters
mod.execParams = "DATFILE='jraw:',DBDATA='folio(ind,ret,ris,na)' + "DBSOL='folioresult(ind,val)'");
mod.run(); // Run the model

Advanced data handling features
More I/O driver examples

• Duplicating model output

```c
fopen("tee:portfoliout.txt", F_OUTPUT+F_APPEND)
writeln("Solution:\n Total return: ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, ",")
fclose(F_OUTPUT)
```

• Combining I/O drivers

```c
initializations from "bin:zip:rmt:foliodata.zip"
[RET,RISK,NA] as DBDATA
end-initializations
```

• Use of extended file names

```c
res:= compile("", "portfolio.mos", "tmp:portfolio.bim")
load(modPortf, "tmp:portfolio.bim")
```

4.2 XML interface

**mmxml: XML handling**

- The module *mmxml* provides an XML interface for the Mosel language.
  - **New type:** `xmldoc` represents an XML document
  - Each node/element in the document is identified by a **node number** (an integer) that is attached to the document (i.e., a node number cannot be shared by different documents)
- Documentation: Chapter *mmxml* in the *Mosel Language Reference*

**Structure of an XML document**

```xml
<?xml ... ?>  Preamble
<root>
  <parent>
    <element attrname="attrvalue"> contents
      <child>
        <leaf>leafcontents</leaf>
      </child>
      <child>2nd child contents</child>
    </element>
    <emptyelement attrname="attrvalue" />
  </parent>
</root>
```

**Example: Reading XML format data**

- **Data file** `folio.xml`:

```xml
<portfolio>
  <share name="treasury" ret="5" dev="0.1" country="Canada"
        region="NA" risk="low />
  <share name="hardware" ret="17" dev="19" country="USA"
        region="NA" risk="high />
  ...
  <share name="electronics" ret="21" dev="16" country="Japan"
        region="Asia" risk="high />
</portfolio>
```
Example: Reading XML format data

```plaintext
declarations
SHARES: set of string  ! Set of shares
RISK: set of string    ! Set of high-risk values among shares
RET: array(SHARES) of real ! Estimated return in investment
AllData: xmldoc        ! XML document
NodeList: list of integer ! List of XML nodes
end-declarations

! Reading data from an XML file
load(AllData, DATAFILE)
getnodes(AllData, "portfolio/share", NodeList)
RISK:= union(l in NodeList | getattr(AllData,l, "risk")="high")
{getstrattr(AllData,l, "name")}
forall(l in NodeList)
RET(getstrattr(AllData,l, "name")):= getintattr(AllData, l, "ret")
```

Example: Outputting data in XML format

```plaintext
declarations
SHARES: set of string  ! Set of shares
frac: array(SHARES) of mpvar ! Fraction of capital used per share
ResData: xmldoc        ! XML document
Share,Root,Sol: integer  ! XML nodes
end-declarations

! Create solution representation in XML format
Root:=addnode(ResData, 0, XML_ELT, "result") ! Create root node
Sol:= addnode(ResData, Root, XML_ELT, "solution") ! Add "solution" node
forall(s in SHARES)
Share:=addnode(ResData, Sol, XML_ELT, "share") ! Add node to "solution"
setattr(ResData, Share, "name", s) ! ...with attr. "name"
setvalue(ResData, Share, frac(s).sol) ! ...and solution value
end-do
save(ResData, "result.xml") ! Save solution to XML format file
save(ResData, Sol, "")      ! Display XML format solution on screen
```

• Generated output file result.xml:

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<result>
  <solution>
    <share name="treasury">0.3</share>
    <share name="hardware">0</share>
    ...
    <share name="electronics">0</share>
  </solution>
</result>
```

Example: XML paths

```plaintext
declarations
DB: xmldoc
Employees, AllEmployees, Names: list of integer
end-declarations

! Check a property of a text node (start from document root)
getnodes(DB, "persList/employee/language[position()=3]/..", Employees)
forall(p in Employees) save(DB, p, "")

! Check a property of a text node (start path from a node)
getnodes(DB, "persList/employee", AllEmployees)
forall(n in AllEmployees)
getnodes(DB, n, "./name[starts-with(string()),'J']/../", Employees)

! Check existence of an attribute
getnodes(DB, "persList/employee[@parttime]", Employees)
writeln("Number of part-time workers: ", Employees.size)

! Check a specific attribute value
writeln("Employee with id=T345: ",
  getvalue(DB, getnode(DB, "persList/employee[@id='T345']/name"))
```

Advanced data handling features ©Copyright 2013 Fair Isaac Corporation. All rights reserved. page 12
5 Structuring optimization applications

5.1 Alternative user interfaces

Application architecture

- Single, configurable model file
- Different interfaces for model execution
  - stand-alone mode (command line or through Xpress-IVE) for development
  - graphical interface to configure model and data scenarios for analysis and simulation
  - Java application for running large batches of model instances

Optimization application in Mosel

**Standalone**

```
<table>
<thead>
<tr>
<th>Data files</th>
<th>Mosel model</th>
<th>Output files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**GUI**

```
<table>
<thead>
<tr>
<th>Configuration file</th>
<th>Summary output</th>
<th>Data files</th>
<th>Mosel model</th>
<th>Output files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Embedded into host application**

```
<table>
<thead>
<tr>
<th>Data files</th>
<th>Summary output</th>
<th>Mosel model</th>
<th>Output files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

Structuring optimization applications ©Copyright 2013 Fair Isaac Corporation. All rights reserved. | page 13
Some highlights

- **Model:**
  - easy maintenance through single model
  - deployment as BIM file: no changes to model by end-user
  - language extensions according to specific needs

- **Interfaces:**
  - several run modes adapted to different types of usages
  - efficient data exchange with host application through memory
  - parallel model runs (Java) or repeated sequential runs (GUI)

5.2 Structuring Mosel models

**Structuring Mosel models: File inclusion**

```mosel
model "FixBV"
uses "mmxprs"
include "fixbv_defs.mos"
include "fixbv_pb.mos"
include "fixbv_solve.mos"
solution:=solveprob
printsol(solution)
end-model
```

**Structuring Mosel models: Subroutines**

- **Subroutines** have a similar structure as models (keyword `model` is replaced by `procedure` or `function`)
  - can use local declarations and overloading

```mosel
function solveprob: real
  maximize(Return)
  returned:= getobjval
end-function

procedure printsol(r: real)
  writeln("Total return: ", r)
  forall(s in SHARES | getsol(frac(s))>0)
    writeln(s, "'s ", getsol(frac(s))*100, "% (", getsol(buy(s)), ")")
end-procedure
```
Structuring Mosel models: Packages

- **Package** = library written in the Mosel language
  - making parts of Mosel models re-usable
  - deployment of Mosel code whilst protecting your intellectual property
  - similar structure as models (keyword `model` is replaced by `package`), compiled in the same way
  - included with the `uses` statement
  - definition of new types, subroutines, symbols

```mosel
package "folioutil"

public declarations
SHARES: set of string     ! Set of shares
frac, buy: array(SHARES) of mpvar ! Decision variables
end-declarations

public procedure printsol(r: real)
writeln("Total return: ", r)
forall(s in SHARES | getsol(frac(s))>0)
writeln(s, " : ", getsol(frac(s))*100, "% (", getsol(buy(s)), ")")
end-procedure

definition of new types, subroutines, symbols

end-package
```

Structuring Mosel models: More package examples

```mosel

!**** Add a file to a compressed TAR archive ****
public procedure addtoarchive(archive: string, toadd: string)
declarations
lsf: list of text
end-declarations

makendir(gettmpdir+/"tartemp") ! Create temp. dir.
untar("zlib.gzip:"+archive, gettmpdir+/"tartemp") ! Open archive
findfiles(SYS_RECURS, lsf, gettmpdir+/"tartemp", ".") ! Rebuild archive
newtar(0, "+archive, gettmpdir+/"tartemp", lsf)
removedir(gettmpdir+/"tartemp") ! Cleaning up
end-procedure

!**** Get the number of the calendar week for a date ****
public function getweek(d:date): integer
firstday:=date(getyear(d-getweekday(d)+4), 1, 3)
returned:= (getasnumber(d) - getasnumber(firstday) +
getweekday(firstday+1) + 5) div 7
end-function
```

Structuring Mosel models: Modules

- **Module** = dynamic library written in C
  - modules of the Mosel distribution:
    + solver interfaces: Xpress-Optimizer (LP, MIP, QP), Xpress-Nonlinear, CP
    + database access: ODBC, OCI
    + system commands; model handling; graphics
  - write your own modules for
    + connecting to external software
    + time-critical tasks
    + defining new types, subroutines, operators, I/O drivers, control parameters, symbols
Some highlights of module features

- Interaction with external programs during their execution (callback functions)
  - progress logs
  - user stopping criteria
  - "life" processing of intermediate results

Module mmxprs: Using callback functions

```mosel
uses "mmxprs"

public procedure printsol
  writeln("Total return: ", getsol(Return))
  forall(s in SHARES | getsol(frac(s))>0)
    writeln(s, ": ", getsol(frac(s))*100, ", ", getsol(buy(s)), "]")
end-procedure

setcallback(XPRS_CB_INTSOL, "printsol")
maximize(Return)
```

5.3 Distributed and remote computing

Schemes of decomposition and concurrent solving

- Simple parallel runs
  - different data instances
  - different algorithm configurations
- Decomposition
  - Benders
  - Dantzig-Wolffe
- Column generation
  - loop over top node
  - branch-and-price
- Cut generation
  - (cut-and-branch, branch-and-cut)
  - adding constraints

Schemes of parallelization

- Simple submodel run
• Iterative sequential submodel runs (decomposition algorithms)

- Communicating concurrent submodels

**Distributed model execution**

- **Module mmjobs:**
  - load several models in memory and execute them concurrently
  - synchronization mechanism based on event queues
  - data exchange between concurrent models through shared memory or memory pipes

- **New:** extending capacities for handling multiple models to distributed computing using several *Mosel instances* (running locally or on remote nodes connected through a network)

- Remote machine must run a server
  - Default: Mosel server *xprmsrv* (started as separate program, available for all platforms supported by Xpress), connect with driver \texttt{xsrv}
    \begin{verbatim}
    connect(mosInst, "ABCD123")
    ! Same as: connect(mosInst, "xsrv:ABCD123")
    \end{verbatim}
  - Alternative: other servers, connect with driver *rcmd*, e.g. with *rsh*, (NB: Mosel command line option \texttt{-r} is required for remote runs):
    \begin{verbatim}
    connect(mosInst, "rcmd:rsh ABCD123 mosel -r")
    \end{verbatim}
Executing a submodel

```mosel
model "Run model rtparams"
uses "mmjobs"
declarations
modPar: Model
end-declarations
! Compile the model file
if compile("rtparams.mos")<>0 then exit(1); end-if
! Load the bim file
load(modPar, "rtparams.bim")
! Start model execution + parameter settings
run(modPar, "PARAM1="+3.4+"",PARAM3='a string" + ",PARAM4="+true)
wait！Wait for model termination
dropnextevent！Ignore termination event message
end-model
```

Executing a submodel remotely

```mosel
model "Run model rtparams remotely"
uses "mmjobs"
declarations
modPar: Model
mosInst: Mosel
end-declarations
NODENAME:= ""! "" for current node, or name, or IP address
! Open connection to a remote node
if connect(mosInst, NODENAME)<>0 then exit(2); end-if
! Load the bim file
load(mosInst, modPar, "rmt:rtparams.bim")
! Start model execution + parameter settings
run(modPar, "PARAM1="+3.4+"",PARAM3='a string" + ",PARAM4="+true)
wait！Wait for model termination
dropnextevent！Ignore termination event message
end-model
```

```mosel
model "Compile and run model rtparams remotely"
uses "mmjobs"
declarations
modPar: Model
mosInst: Mosel
end-declarations
NODENAME:= ""! "" for current node, or name, or IP address
! Open connection to a remote node
if connect(mosInst, NODENAME)<>0 then exit(2); end-if
if compile(mosInst, "", "rmt:rtparams.mos", "rtparams.bim")<>0 then exit(1); end-if
! Load the bim file
load(mosInst, modPar, "rtparams.bim")
! Start model execution + parameter settings
run(modPar, "PARAM1="+3.4+"",PARAM3='a string" + ",PARAM4="+true)
wait！Wait for model termination
dropnextevent！Ignore termination event message
end-model
```

Distributed solving

- Use all the available computing power by solving (sub)models on physical or virtual remote machines

- Physical location of model files, input and result data depending on application
**XPRD**

- *Mosel remote invocation library*
- Build applications requiring the Xpress technology that run from environments where Xpress is not installed
- Relies on the *Mosel Distributed Framework* (see Mosel module *mmjobs*)
- Self-contained library (no dependency on the usual Xpress libraries)

**Remote model run: mmjobs**

```mosel
uses "mmjobs"

declarations
mosInst: Mosel
modRP: Model
end-declarations

NODENAME := ""  ! IP address, or "" for current node

if connect(mosInst, NODENAME) <> 0 then exit(2); end-if
if compile(mosInst, "rmt:rtparams.mos", "tmp:rp.bim") <> 0 then exit(1); end-if
load(mosInst, modRP, "tmp:rp.bim") ! Load bim file into remote instance
run(modRP, "PARAM1=" + 2 + ",PARAM2=" + 3.4 + ",PARAM3='a string'" + ",PARAM4=true")
wait ! Wait for model termination
dropnextevent ! Ignore termination event message
writeln("'rtparams' returned: ", getexitcode(modRP))
disconnect(mosInst) ! Disconnect remote instance
```

**Remote model run: XPRD**

```java
public static void main(String[] args) throws Exception {
    XPRD xprd=new XPRD();  // Initialize XPRD
    XPRDMosel mosInst=null;
    XPRDModel modRP=null;
    String NODENAME = "";  // IP address, or "" for current node
    mosInst=xprd.connect(NODENAME);  // Open connection to a remote node
    mosInst.compile("", "rmt:rtparams.mos", "tmp:rp.bim");  // Compile the model file
    modRP=mosInst.loadModel("tmp:rp.bim");
    modRP.execParams = "PARAM1=" + 2 + ",PARAM2=" + 3.4 + " ,PARAM3='a string'" + ",PARAM4=true");
    modRP.run();  // Run the model
    xprd.waitForEvent();  // Wait for model termination
    xprd.dropNextEvent();  // Ignore termination event message
    System.out.println("'rtparams' returned: "+ modRP.getResult());
    mosInst.disconnect();  // Disconnect remote instance
}
```

**Solvers and solution algorithms**

### 6.1 Switching and combining solvers

**Example: Chess problem**

- A joinery makes different sizes of boxwood chess sets. Each size has a specific requirement of machining time and wood, and yields a given profit.
- There are 4 lathes with skilled operators who each work a 40 hour week. Only 200 kg of boxwood can be obtained per week.
- How many sets of each kind should be made each week so as to maximize profit?
Chess problem: Mathematical formulation

- $x_i$ – quantity of chess sets of size $i$ made

Maximize $\sum_{i \in \text{PRODS}} \text{PROFIT}_i \cdot x_i$

subject to

- $\sum_{i \in \text{PRODS}} \text{DUR}_i \cdot x_i \leq 160 (= 4 \cdot 40)$ (lathe time)
- $\sum_{i \in \text{PRODS}} \text{WOOD}_i \cdot x_i \leq 200$ (wood)
- $\forall x_i$ integer

Chess problem: Implementation

```plaintext
model "Chess (MIP)"
uses "mmxprs"

declarations
PRODS = 1..NP ! Index set
DUR, WOOD, PROFIT: array(PRODS) of real ! Input data
x: array(PRODS) of mpvar ! Production quantities
end-declarations

initializations from "chess.dat"
DUR WOOD PROFIT
end-initializations

sum(i in PRODS) Dur(i)*x(i) <= 160 ! Constraint definition
sum(i in PRODS) Wood(i)*x(i) <= 200
forall(i in PRODS) x(i) is_integer

MaxProfit:= sum(i in PRODS) Profit(i)*x(i) ! Objective function
maximize(MaxProfit)
writeln("Solution: ", MaxProfit.sol)
end-model
```

Chess problem: Extension

- What happens if...
  ...board size becomes a variable and
  ...coefficients are defined by these functions of the size?

$\text{Profit}_i = \frac{9}{64} \cdot \text{size}_x^3 + \frac{167}{16} \cdot \text{size}_x - 2$

$\text{Dur}_i = 1.5 + \frac{1}{\text{size}_x}$

$\text{Wood}_i = \frac{3}{8} \cdot \text{size}_x^2 + \frac{1}{2} \cdot \text{size}_x + \frac{1}{2}$

Chess problem: Implementation (NLP)

```plaintext
model "Chess (NL)"
uses "mmxnlp"

declarations
PRODS = 1..NP ! Index set
Dur, Wood, Profit: array(PRODS) of nlctr ! "Coefficient" expressions
x: array(PRODS) of mpvar ! Production quantities
sizex: array(PRODS) of mpvar ! Size variables
end-declarations

forall(i in PRODS) do
  Dur(i) : = 1.5 + 1/size(i)
  Wood(i) : = 3/8*size(i)^2 + 1/2*size(i) + 1/2
  Profit(i) : = 9/64*size(i)^3 + 167/16*size(i) - 2
  0.5 <= sizex(i); sizex(i) <= 5
end-do

forall(i in PRODS | i>1) sizex(i)>=sizex(i-1)+1 ! Different sizes

sum(i in PRODS) Dur(i)*x(i) <= 160 ! Constraint definition
sum(i in PRODS) Wood(i)*x(i) <= 200
forall(i in PRODS) x(i) is_integer

MaxProfit:= sum(i in PRODS) Profit(i)*x(i) ! Objective function
```

---

Solvers and solution algorithms ©Copyright 2013 Fair Isaac Corporation. All rights reserved.
maximize(MaxProfit)  ! Solve the problem
writeln("Solution: ", MaxProfit.sol)
forall(i in PRODS)
    writeln("Size ", sizex(i).sol, ": ", x(i).sol, ", p": , Profit(i).sol,
        " d":, Dur(i).sol, " w":, Wood(i).sol)
end-model

- Use known (MIP solution) values as initial values for NLP

declarations
    DUR, WOOD, PROFIT: array(PRODS) of real  ! MIP Coefficients
    startsol: array(mpvar) of real  ! Solution values
end-declarations

***** Solve subproblems to determine start values *****
calc_startvalues_x  ! Solve MIP problem
calc_startvalues_size  ! Solve NL problem
forall(i in PRODS) setinitval(sizex(i), startsol(sizex(i)))
forall(i in PRODS) setinitval(x(i), startsol(x(i)))

***** MIP problem: Using known values for coefficients *****
procedure calc_startvalues_x
    initializations from DATAFILE  ! Read data from file
    DUR WOOD PROFIT
end-initializations
with mpproblem do
    sum(i in PRODS) DUR(i)*x(i) <= 160  ! Constraint definition
    sum(i in PRODS) WOOD(i)*x(i) <= 200
    forall(i in PRODS) x(i) is_integer
    maximize(sum(i in PRODS) PROFIT(i)*x(i))
    if getprobstat=XPRS_OPT then
        writeln("MIP Solution: ", getobjval)
        savemipsol(startsol)
    end-if
end-do
end-procedure

***** CP problem: Calculate x(i) for f(x(i))=r(i)*****
procedure calc_startvalues_size
    declarations
        s: array(PRODS) of cpfloatvar
    end-declarations
    forall(i in PRODS) do
        LB <= s(i); s(i) <= UB
    end-do
    forall(i in PRODS) WOOD(i) = 3/8*s(i)^2 + 1/2*s(i) + 1/2
    if cp_find_next_sol then
        forall(i in PRODS) writeln(WOOD(i), ": ", s(i).sol)
    end-if
end-procedure

- Multiple optimization problems can be defined within a single optimization model, such problems can share data, and make use of common decision variables
6.2 Multiple models interacting via callbacks

Example: Machine assignment and sequencing

- We need to produce different products on a set of machines. Each machine may produce all of the products but processing times and costs vary. For every product we are given its release and due dates.
- We wish to determine a production plan for all products that minimizes the total production cost.

Assignment and sequencing: Mathematical model

- We can represent this problem by two subproblems:
  1. the machine assignment problem (implemented by a MIP model)
  2. the sequencing of operations on machines (formulated as a CP single machine problem)

Assignment and sequencing: Algorithm

- Idea: at the nodes of a MIP Branch-and-Bound search, solve CP subproblems for generating no-good cuts if the set of tasks assigned to a machine cannot be scheduled

Assignment and sequencing: Implementation

- Multiple optimization problems implemented as separate model (files) make parallel and multithreaded optimization easily accessible

```mosel
declarations
  PRODS = 1..NP   ! Set of products
  MACH = 1..NM    ! Set of machines
  REL, DUE: array(PRODS) of integer    ! Release, due dates of orders
  COST, DUR: array(PRODS, MACH) of integer ! Processing cost, times
  use: array(PRODS, MACH) of mpvar    ! 1 if p uses m, otherwise 0
  Cost: linctr ! Objective function
end-declarations

!*** MIP master model ***
! Objective: total processing cost
Cost:= sum(p in PRODS, m in MACH) COST(p,m) * use(p,m)

! Each order needs exactly one machine for processing
forall(p in PRODS) sum(m in MACH) use(p,m) = 1
forall(p in PRODS, m in MACH) use(p,m) is_binary
```

Solvers and solution algorithms ©Copyright 2013 Fair Isaac Corporation. All rights reserved.
Valid inequalities for strengthening the LP relaxation

\[
\text{MAX\_LOAD} := \max_{p \in \text{PRODS}} \text{DUE}(p) - \min_{p \in \text{PRODS}} \text{REL}(p)
\]

forall \(m \in \text{MACH}\) \(\sum_{p \in \text{PRODS}} \text{DUR}(p, m) \times \text{use}(p, m) \leq \text{MAX\_LOAD}\)

setcallback(XPRS_CB_CUTMGR, "generate_cuts")  \(\text{! Define cut manager cb.}\)
minimize(Cost)  \(\text{! Solve the problem}\)

```java
Public function generate_cuts: boolean
returned:=false; ctcutold:=ctcut
forall (m in MACH) do
    if generate_cut_machine(m) then
        returned:=true  \(\text{! Call func. again for this node}\)
        ctcut+=1
    end-if
end-do
if returned then
    writeln("Node ", getparam("XPRS_NODES"), ": ",
            ctcut-ctcutold, ", cut(s) added")
end-if
end-function
```

```java
Function generate_cut_machine(m: integer): boolean
declarations
    ProdMach: set of integer
end-declarations
! Collect the operations assigned to machine m
    products_on_machine(m, ProdMach)
! Solve the sequencing problem (CP model): if solved, save the solution,
! otherwise add an infeasibility cut to the MIP problem
    size:= getsize(ProdMach); returned:=false
    if (size>1) then
        if not solve_CP_problem(m, ProdMach, 1) then
            Cut:= \(\sum_{p \in \text{ProdMach}} \text{use}(p, m) - (\text{size}-1)\)
            addcut(1, CT_LEQ, Cut)
            returned:=true
        end-if
    end-if
end-function
```

Assignment and sequencing: Extension

- The sequencing subproblems are independent and could therefore be solved concurrently
  \(\Rightarrow\) requires reformulation to the master model to coordinate parallel run; no changes to subproblem(s)

6.3 Distributed computing and metaheuristics

Example: TSP (Traveling Salesman Problem)

- Determine the tour of shortest length (least cost) that visits every location from a given set exactly once.

TSP: Mathematical model

- Objective: minimize total distance
  \[
  \min \sum_{i,j \in \text{NODES}} \text{DIST}_{ij} \cdot \text{fly}_{ij}
  \]
- Variables:
  \(\forall i, j \in \text{NODES} : \text{fly}_{ij} \in \{0, 1\}\)
- Visit every location once:
  \(\forall i \in \text{NODES} : \sum_{j \in \text{NODES}} \text{fly}_{ij} = 1\)
  \(\forall j \in \text{NODES} : \sum_{i \in \text{NODES}} \text{fly}_{ij} = 1\)
- Need to add subtour breaking constraints or iterative subtour elimination
TSP: Algorithm

- **Idea**: generate a heuristic solution by combining tours from regional subproblems
  - solve small subproblems of neighboring nodes (belonging to the same ‘region’)
  - ‘glue’ pairs of neighboring regions by unfixing arcs close to their common border and re-solving the resulting problem
  - iteratively, extend to the whole set of locations

  ⇒

- **Subproblems can be solved independently**
  ⇒ concurrent solving with several nodes
  - determine a precedence tree of (sub)problems to solve
  - a subproblem is added to the job queue once both its predecessors have been solved
  - whenever a node becomes available, send it the next job

TSP: Implementation

- **Remote instance**
- **Submodel**
- **Local instance**
- **Master model**

  !************ Formulate and solve a TSP (sub)problem ************

  declarations
  DIST: array(NodeSet,NodeSet) of real ! Distance between cities
  NEXTC: array(NodeSet) of integer ! Next city after i in solution
  fly: array(NodeSet,NodeSet) of mpvar ! 1 if flight from i to j
  end-declarations

  ! Visit every city once
  forall(i in NodeSet) sum(j in NodeSet | i<>j) fly(i,j) = 1
  forall(j in NodeSet) sum(i in NodeSet | i<>j) fly(i,j) = 1
  forall(i,j in NodeSet | i<>j) fly(i,j) is_binary

  ! Fix part of the variables
  forall(i in FixedSet | SOL(i) not in UnfixedSet) fly(i,SOL(i)) = 1

  ! Objective: total distance
  TotalDist:= sum(i,j in NodeSet | i<>j) DIST(i,j)*fly(i,j)
  minimize(TotalDist)
  if LEVEL=1 then two_opt; end-if ! 2-opt for partially fixed prob.s

  !************ Implementation of job queue handling ************

  declarations
  modPar: array(RM) of Model ! Models
  Msg: Event ! Messages sent by models
  modid: array(set of integer) of integer ! Model index for model IDs

Solvers and solution algorithms ©Copyright 2013 Fair Isaac Corporation. All rights reserved. page 24
jobid: array(set of integer) of integer  ! Job index for model IDs
JobList: list of integer  ! List of jobs
JobsRun: set of integer  ! Set of finished jobs
JobSize: integer  ! Number of jobs to be executed

end-declarations

JobList:= sum(i in JOBS) [i]  ! Define the list of jobs (instances)
JobSize:= JobList.size  ! Store the number of jobs
JobsRun:= {}  ! Set of terminated jobs is empty

**** Start initial lot of model runs ****
forall (m in RM)
  if JobList<>[] then
    start_next_job(m)
  end-if

**** Run all remaining jobs ****
while (JobsRun.size<JobSize) do
  wait  ! Wait for model termination
  Msg:= getnextevent
  if getclass(Msg)=EVENT_END then  ! We are only interested in "end" ev.
    m:=getfromid(Msg)
    JobsRun+={jobid(m)}  ! Keep track of job termination
    if JobList<>[] then  ! Start a new run if queue not empty
      start_next_job(m)
    end-if
  end-if
end-do

**** Start next job ****
procedure start_next_job(m:integer)
  i:=getfirst(JobList)  ! Retrieve first job in the list
  cuthead(JobList,1)  ! Remove first entry from job list
  jobid(getid(modPar(modid(m)))):= i
  run(modPar(modid(m)), "PB=" + i + ",LEVEL=" + LEV(i) + ",NUM=" + n)
end-procedure

Summary

• Have seen:
  – design choices for optimization applications
    (target audience, project design, algorithms)

• Xpress-Mosel:
  – recent developments make possible implementation of complex algorithms and a
    high degree of user interaction
  – unique features for handling large-scale problems:
    support of decomposition, concurrent solving, distributed computing, and also
    64bit coefficient indexing

Where to get more information

• Xpress website:
  – http://www.fico.com/xpress

• Xpress resources (documentation, whitepapers)
  – http://optimization.fico.com

• Searchable on-line examples database:
  – http://examples.xpress.fico.com

• Trial download:
  – http://decisions.fico.com/downloadTrial.html
Next talks

- *Follow-up talk:*
  **Wednesday 3 July, 8:30-10:00, WA-9, room 03-3**
  Oliver Bastert: Designing interactive user interfaces for optimization models

- *FICO Xpress post-conference workshop:*
  **Thursday 4 July 11:00-18:00, room R19-2**
  - 11:00-13:00 Introduction to working with Mosel
  - 14:00-18:00 New features of Xpress Release 7.5

⇒ *Please sign up at the FICO booth*