Xpress-Mosel

Implementing decomposition approaches for concurrent and distributed solving

Susanne Heipcke
Xpress Team, FICO
http://www.fico.com/xpress
Overview

» Overview of Xpress
» Xpress-Mosel
» Examples
» Summary
How to deal with problems when they grow too large

» Buy a faster machine
How to deal with problems when they grow too large

» Buy a faster machine
» Get a better/newer solver
How to deal with problems when they grow too large

» Buy a faster machine
» Get a better/newer solver
» Tune your solver
How to deal with problems when they grow too large

- Buy a faster machine
- Get a better/newer solver
- Tune your solver
- Work on your model
How to deal with problems when they grow too large

» Buy a faster machine
» Get a better/newer solver
» Tune your solver
» Work on your model
  » simplify/aggregate
  » decompose
  » sequential
  » parallel
» choice of solver/solution method
Decomposition is the process of breaking large optimization problems into smaller, more manageable sub-problems and solving them either sequentially or in parallel.
Overview of Xpress
Overview of Xpress

» Optimization algorithms
  » solve different classes of problems
  » built for speed, robustness and scalability
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
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
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
### Overview of Xpress

<table>
<thead>
<tr>
<th>GUI (development/deployment)</th>
<th>IVE</th>
<th>Insight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Mosel language</td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td>Mosel modules</td>
<td></td>
</tr>
<tr>
<td>Solvers/data/applications</td>
<td>Optimizer (LP/MIP/QP)</td>
<td>SLP Knitro</td>
</tr>
<tr>
<td></td>
<td>BCL</td>
<td>External data sources</td>
</tr>
<tr>
<td></td>
<td>mmxprs</td>
<td>mmxnlp</td>
</tr>
</tbody>
</table>
Xpress innovations

» Solving
  1983: LP solver running on PCs
  1992: parallel MIP (1997 on distributed PC/Linux networks)
  1995/1996: commercial branch and cut algorithm
  1998: bound switching in dual simplex
  2003: lift-and-project cuts
  2009: parallel MIP heuristics
  2010: LP/MIP solver crosses 64-bit coefficient indexing threshold
  2013: automatic solver selection for NLP

» Modeling
  1983: general purpose algebraic modeling language (mp-model)
  2001: algebraic modeling language combining modeling, solving, and programming (Mosel)
  2005: profiler and debugger for a modeling language
  2005: user-controlled parallelism at the model level
  2010: modeling language supporting distributed computing
  2012: remote invocation without local installation (XPRD)
» A modeling and solving environment
  » integration of modeling, solving, and programming facilities
    ⇒ 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

» Language is concise, user friendly, high level
  ⇒ rapid development and deployment
» Mosel language: to implement problems and solution algorithms
⇒ model or Mosel program
» platform-independent compiled models for distribution to protect intellectual property
» Mosel language: to implement problems and solution algorithms
⇒ *model* or *Mosel program*
» platform-independent compiled models for distribution to protect intellectual property

» Mosel Model Compiler and Run-time Libraries: to compile, execute and access models from a programming language
⇒ *C/C++, C#, Java, or VB program*
Model example

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

declarations
SHARES = 1..10           ! Set of shares
RISK = {2,3,4,9,10}      ! Set of high-risk values among 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]

! Objective: total return
Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
sum(s in RISK) frac(s) <= 1/3

! Minimum amount of North-American values
sum(s in NA) frac(s) >= 0.5

! Spend all the capital
sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
forall(s in SHARES) frac(s) <= 0.3

! Solve the problem
maximize(Return)

! Solution printing
writeln("Total return: ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "/")
end-model
» Mosel Native Interface (NI): to provide new or extend existing functionality of the Mosel language ⇒ module (DSO)

Modules of the Mosel distribution:
» solvers: mmxpfrs, mmquad, mmxnlp, mmnl, kalis
» data I/O: mmetc, mmmodbc, mmoci, mmsheet, mmxml
» model handling, utilities: mmjobs, mmsystem
» graphics, GUI: mmive, mmxad, mminsight
Mosel: Components and interfaces

» Mosel Native Interface (NI): to provide new or extend existing functionality of the Mosel language ⇒ module (DSO)

Modules of the Mosel distribution:
» solvers: mmxprs, mmquad, mmxnlp, mmnl, kalis
» data I/O: mmetc, mmodbc, mmoci, mmsheet, mmxml
» model handling, utilities: mmjobs, mmsystem
» graphics, GUI: mmive, mmxad, mminsight

» Xpress-IVE: graphical user interface
» representation of the problem matrix, solution status/progress graphs, and result display

» Tools: debugger, profiler
Mosel modules

**Mosel**

- **Solvers**
  - MISLP
  - MIQP
  - QP
  - QCQP
  - MI
  - LP
  - SLP

- **User extension**
  - ODBC, OCI, XML, CSV, ...
  - mmjobs

- **System, extension**
  - Kalis

- **GUI**
  - Insight

- **Data interfaces**
  - XAD
  - IVE

- **Extensions**
  - User extension
  - System, mmjobs

© 2013 Fair Isaac Corporation.
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
Executing a submodel

» Basic submodel execution:

User

Master

compile/load/run submodel

wait for termination

process results

Submodel

start

results

results
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 for model termination
wait

! Ignore termination event message
dropnextevent

end-model
Executing a submodel remotely

model "Run model rtparams remotely"
  uses "mmjobs"

declarations
  modPar: Model
  mosInst: Mosel
end-declarations

! Compile the model file
if compile("rtparams.mos")<>0 then exit(1); end-if

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(mosInst, modPar, "rmt:rtparams.bim")
  ! Load the bim file
run(modPar, "PARAM1=" + 3.4 + ",PARAM3='a string’" + ",PARAM4=" + true)
  ! Start model execution + parameter settings
wait
  ! Wait for model termination
dropnextevent
  ! Ignore termination event message
end-model
» Mosel remote invocation library

» Build applications requiring the Xpress technology that run from environments where Xpress is not installed
» 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*)
» 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)
uses "mmjobs"

declarations
  mosInst: Mosel
  modRP: Model
end-declarations

NODENAME:= "" ! IP address, or "" for current node
                   ! Open connection to a remote 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 ! Compile the model file remotely
load(mosInst, modRP, "tmp:rp.bim") ! Load bim file into remote instance
                   ! Start model execution
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
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
                          // Open connection to a remote node
    mosInst=xprd.connect(NODENAME);
                          // Compile the model file
    mosInst.compile("", "rmt:rtparams.mos", "tmp:rp.bim");
                          // Load bim file into remote instance
    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
}
Examples

» Multiple problems and solution loading
» Distributed computing and metaheuristics
Example: Jobshop scheduling

» Schedule the production of a set of jobs on a set of machines. Every job is produced by a sequence of tasks, each of these tasks is processed on a different machine. A machine processes at most one job at a time. Tasks are non-preemptive (once started the cannot be interrupted).
Jobshop: Mathematical model

» Objective: minimize makespan

» Variables:
\[ \forall j \in \text{JOBS}, t \in \text{TASKS} : \text{start}_{jt}, \text{comp}_{jt} \in [0, \text{MAXTIME}] \]
\[ \forall m \in \text{RESOURCES}, i < j \in \text{JOBS} : y_{mij} \in \{0, 1\} \]

» Precedence relations:
\[ \forall j \in \text{JOBS} : \text{makespan} \geq \text{start}_{j,\text{NBRES}} + \text{DUR}_{j,\text{NBRES}} \]
\[ \forall j \in \text{JOBS}, t \in \text{TASKS} - \{\text{NBRES}\} : \text{start}_{jt} + \text{DUR}_{jt} \leq \text{start}_{j,t+1} \]

» Disjunctions between tasks on same machine:
\[ \forall m \in \text{RESOURCES}, i < j \in \text{JOBS} : \]
\[ \text{start}_{i,\text{RES}_{mi}} + \text{DUR}_{i,\text{RES}_{mi}} \leq \text{start}_{j,\text{RES}_{mj}} + M \cdot y_{mij} \]
\[ \text{start}_{j,\text{RES}_{mj}} + \text{DUR}_{j,\text{RES}_{mj}} \leq \text{start}_{i,\text{RES}_{mi}} + M \cdot (1 - y_{mij}) \]

» Linking start and completion times
\[ \forall j \in \text{JOBS}, t \in \text{TASKS} : \text{start}_{jt} + \text{DUR}_{jt} = \text{comp}_{jt} \]
» **Idea:** by solving single machine subproblems (sequencing the tasks processed on the same machine) we generate partial solutions that are loaded as start solutions into the MIP optimizer.
» Idea: by solving single machine subproblems (sequencing the tasks processed on the same machine) we generate partial solutions that are loaded as start solutions into the MIP optimizer

» State and solve two distinct problems:

1. several instances (one per machine) of a sequencing problem
2. one instance of the jobshop problem
The "multis":
– multi-problem

» Multiple optimization problems can be defined within a single optimization model, such problems can share data, and make use of common decision variables
» Start solution algorithm

declarations
    y: dynamic array(RESOURCES,JOBS,JOBS) of mpvar  ! Disjunction var.s
    heursol: dynamic array(set of mpvar) of real  ! JS start solution
    SeqProblem: mpproblem  ! Opt. problem
    pos: array(JOBS) of integer  ! Sequencing solution values
end-declarations

! Solve the single machine sequencing problems
forall(m in RESOURCES) do
    with SeqProblem do
        sequence_machine(m)  ! Formulate the problem
        minimize(makespan)  ! Solve the problem
        writeln("*** Machine ", m, ": ", getobjval)
        forall(j in JOBS) pos(j) := round(sum(k in JOBS) k*rank(j,k).sol)
    end-do
forall(i, j in JOBS | i<j ) heursol(y(m,i,j)):= if(pos(i)<pos(j), 0, 1)
loadprob(makespan)
addmipsol("mach"+m, heursol)       ! Load the heuristic solution
delcell(heursol)                   ! Delete saved solution values
reset(SeqProblem)                  ! Delete subproblem definition
end-do

setparam("XPRS_HEURSEARCHEFFORT", 2)
setparam("XPRS_HEURSEARCHROOTSELECT", 31)
setparam("XPRS_HEURSEARCHTREESELECT", 19)

! Solve the jobshop problem
minimize(makespan)
The sequencing subproblem

declarations
  makespan: mpvar ! Schedule completion time
  rank: array(JOBS,JOBS) of mpvar ! =1 if job j at position k
  jcomp, tstart: array(JOBS) of mpvar ! Start time of job at position k
end-declarations

procedure sequence_machine(m:integer)
  ! One job per position
  forall(k in JOBS) sum(j in JOBS) rank(j,k) = 1

  ! One position per job
  forall(j in JOBS) sum(k in JOBS) rank(j,k) = 1

  ! Sequence of jobs
  forall(k in 1..NBJOBS-1)
    tstart(k+1) >=
      tstart(k) + sum(j in JOBS) DUR(j,RES(m,j))*rank(j,k)
! Start times (release date = min total duration for preceding tasks)
forall(j in JOBS) REL(j):= \sum(t \text{ in } 1..\text{RES}(m,j)-1) \text{ DUR}(j,t)
forall(j in JOBS) DURSUCC(j):= \sum(t \text{ in } \text{RES}(m,j)+1..\text{NBRES}) \text{ DUR}(j,t)

forall(k in JOBS) tstart(k) \geq \sum(j \text{ in } \text{JOBS}) \text{ REL}(j) \times \text{rank}(j,k)

! Completion times
forall(k in JOBS) jcomp(k) =
  tstart(k) + \sum(j \text{ in } \text{JOBS}) (\text{DUR}(j,\text{RES}(m,j)) + \text{DURSUCC}(j)) \times \text{rank}(j,k)

forall(j,k \text{ in } \text{JOBS}) \text{ rank}(j,k) \text{ is_binary}

! Objective function: latest completion time
forall(k \text{ in } \text{JOBS}) \text{ makespan} \geq \text{jcomp}(k)
end-procedure
The sequencing subproblems are independent and could therefore be formulated and solved concurrently.

⇒ requires implementation of subproblem(s) in a separate model file
Jobshop: Extension

declarations
SeqMod: array(RESOURCES) of Model ! Models
modid: dynamic array(set of integer) of integer ! Model indices
end-declarations

! Compile submodel and prepare submodel data
if compile("jobseq.mos")<>0 then exit(1); end-if
initializations to "bin:shm:jsdata"
  DUR RESIND MAXDUR
end-initializations

! Solve the single machine sequencing problems
forall(m in RESOURCES) do
  load(SeqMod(m), "jobseq.bim")
  modid(getid(SeqMod(m))):= m
  run(SeqMod(m), "MACH="+m +",NBJOBS="+NBJOBS +",NBRES="+NBRES +",DATAFILE=bin:shm:jsdata,MAXTHRD="+1)
end-do
while (tct<NBRES) do

wait  ! Wait for the next event
Msg:=getnextevent  ! Get the event
mid:=modid(getfromid(Msg))  ! Get model number of sender
if getclass(Msg)=NEWSOL then  ! Get the event class

initializations from "bin:shmem:sol"+mid
  pos  Makespan
end-initializations
forall(i,j in JOBS | i<j )
  heursol(y(mid,i,j)):= if(pos(i)<pos(j), 0, 1)
loadprob(makespan)
addmipsol("mach"+mid,heursol)  ! Load the heuristic solution
delcell(heursol)  ! Delete saved solution values
fdelete("bin:shmem:sol"+mid)
ellif getclass(Msg)=NOSOL then
  writeln("Subproblem ", mid, " has no solution. Stopping"); exit(1)
else
  tct+=1  ! Count model terminations
  unload(SeqMod(mid))  ! Delete subproblem
end-if
end-do
The "multis":
– multi-solver
– multi-model

» Multiple optimization problems implemented as separate model (files) make parallel and multithreaded optimization easily accessible
» With subproblems in separate model files, we can easily experiment with alternative problem formulations

» change the submodel file name to switch to a different implementation

⇒ a Constraint Programming formulation (module kalis) using the scheduling solver solves subproblems faster
uses "kalis", "mmsystem"

declarations
  makespan: cpvar ! Schedule completion time
  jcomp: array(JOBS) of cpvar ! Earliest completion times of jobs
  task: array(JOBS) of cptask ! Tasks (jobs to be scheduled)
  res: cpresource ! Resource (machine)
  L: cpvarlist ! List of variables
end-declarations

! Setting up tasks and resource (formulation of the disjunctions)
set_resource_attributes(res, KALIS_UNARY_RESOURCE, >1)
forall(j in JOBS) set_task_attributes(task(j), DUR(j,MACH), res)

! Bounds on start times and schedule length
forall(j in JOBS) do
  REL(j) <= getstart(task(j)); getstart(task(j)) <= MAXTIME
  getend(task(j)) <= MAXTIME
end-do
! Completion times
forall(j in JOBS) DSUCC(j) := sum(t in MACH+1..NBRES) DUR(j,t)
forall(j in JOBS) jcomp(j) = getend(task(j)) + DSUCC(j)

! Objective function: minimize latest completion time
forall(j in JOBS) L += jcomp(j)
make span = maximum(L)

if cp_schedule(makespan) >0 then  ! Solve the problem
   forall(j in JOBS) startsol(j) := getsol(task(j).start)
   qsort(SYS_UP, startsol, sndx)
   forall(j in JOBS) pos(sndx(j)):= j
       ! Save the solution values
   initializations to "bin:shm mem:sol"+MACH
        pos evaluation of makespan.sol as "Makespan"
   end-initializations
   send(NEWSOL, getobjval)  ! Send "solution found" event
else
   send(NOSOL, 0)           ! Send "no solution" event
end-if
Multiple problem handling

Mosel can handle several problems within a model file:

- **Multi-problem** (*mpproblem*): single model file
  - problems share data
  - integrated; no direct access to (sub)problems by other models/applications
  - sequential access to problems

- **Multi-model** (*mmjobs*): several model files
  - communication of data (in memory)
  - stand-alone execution of submodels or use of submodels with other master models/applications
  - sequential or parallel execution of models
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

\[
\text{minimize } \sum_{i,j \in \text{NODES}} \text{DIST}_{ij} \cdot \text{fly}_{ij}
\]

» 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
» **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
The "multis":
- multi-model
- multi-node

» Extension of multiple model handling to distributed computing using several Mosel instances opens new perspectives for the implementation of decomposition approaches
TSP: Implementation

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

! 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) ! Solve the initial problem
break_subtours ! Eliminate subtours
if LEVEL>1 then two_opt; end-if ! 2-opt for partially fixed prob.s
TSP: Implementation

!****************** 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
  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
TSP: Implementation

**** Run all remaining jobs ****
while (JobsRun.size<JobSize) do
  wait
  Msg:= getnextevent
  if getclass(Msg)=EVENT_END then
    m:=getfromid(Msg)
    JobsRun+=\{jobid(m)\}
    if JobList<>[] then
      start_next_job(m)
    end-if
  end-if
end-do

**** Start next job ****
procedure start_next_job(m:integer)
  i:=getfirst(JobList)
  cuthead(JobList,1)
  jobid(getid(modPar(modid(m)))):= i
  run(modPar(modid(m)), "PB=" + i + ",LEVEL=" + LEV(i) + ",NUM=" + n)
end-procedure
» Xpress-Mosel: modeling + solving
  » platform for implementing solution algorithms and complete applications
Summary

» Xpress-Mosel: modeling + solving
  » platform for implementing solution algorithms and complete applications

» Examples of algorithms
  » Job-shop: sequence of MIP problems, addmipsol with partial solutions.
» Xpress-Mosel: modeling + solving
  » platform for implementing solution algorithms and complete applications

» Examples of algorithms
  » Job-shop: sequence of MIP problems, addmipsol with partial solutions.
  
  Extension: concurrent solving using multiple solvers
» Xpress-Mosel: modeling + solving
  » platform for implementing solution algorithms and complete applications

» Examples of algorithms
  » Job-shop: sequence of MIP problems, addmipsol with partial solutions. Extension: concurrent solving using multiple solvers
  » TSP: distributed computing (multiple models), graphics, XPRD, MIP + meta-heuristics
Where to get more information

» Xpress website:
  » http://www.fico.com/xpress

» Xpress documentation:
  » http://optimization.fico.com
     (‘Multiple models and parallel solving with Mosel’ and ‘Hybrid MIP/CP solving with Xpress-Optimizer and Xpress-Kalis’)

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

» Academic Partner Program:
  » http://www.fico.com/app