IntelliFest 2012
International Conference on Reasoning Technologies

INTELLIGENCE IN THE CLOUD

San Diego, Oct-2012

Modeling and Solving Decision Optimization Problems

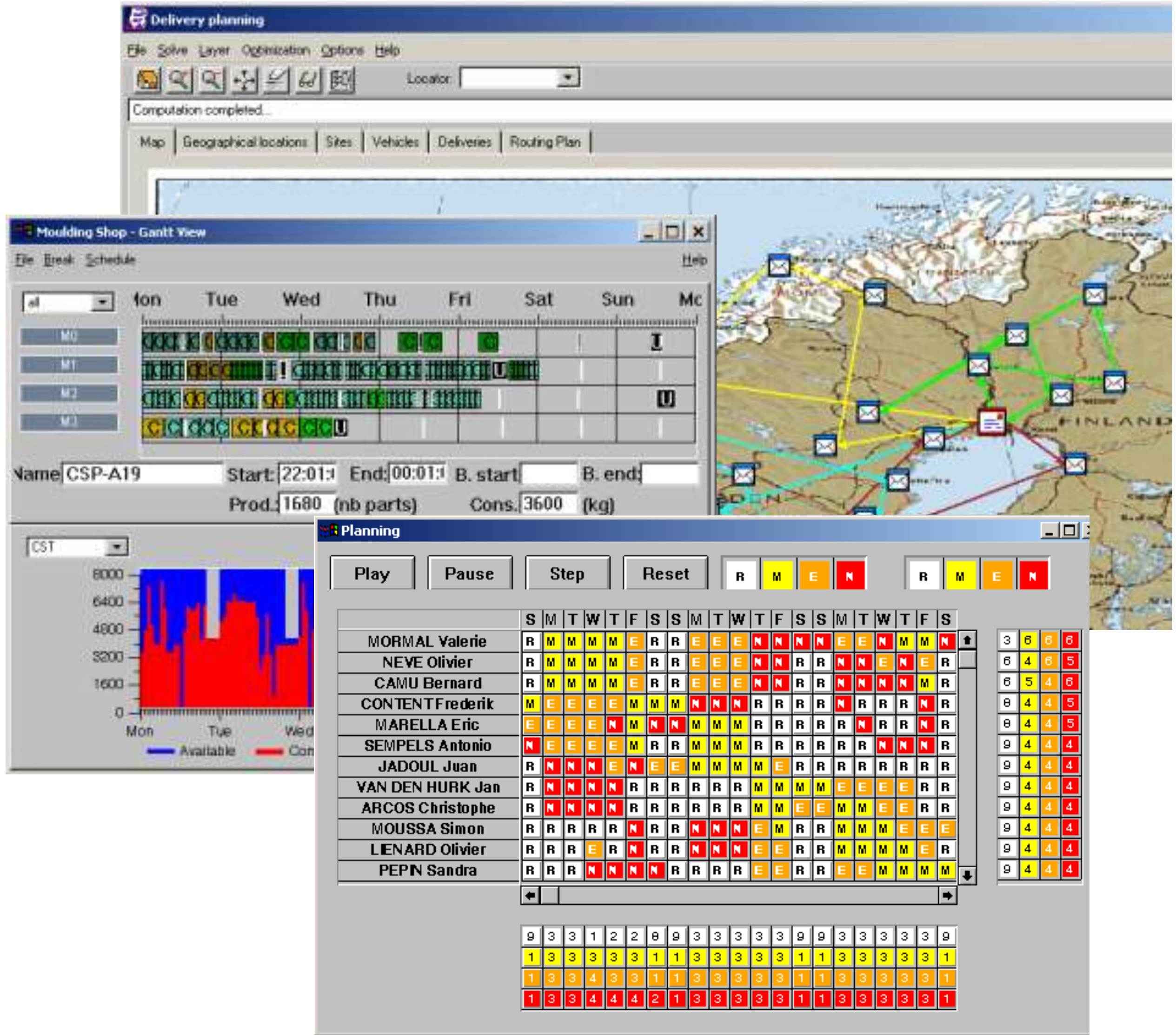
Jacob Feldman, PhD
OpenRules, Inc., CTO
www.openrules.com

Decision Optimization

- Decision Optimization helps *business* people to:
  - make better decisions
  - by finding *optimal* (close to optimal) solutions
  - among multiple alternatives
  - subject to different *business* constraints
- Find the *best* possible resource utilization to achieve a desired optimization objective such as:
  - minimizing expenses or travel time
  - maximizing ROI or service level, etc.
- Relies on proven mathematical techniques such as:
  - Constraint and Linear Programming (CP/LP)
  - integrated in modern decision modeling frameworks

Typical CP Applications

- Scheduling and Resource Allocation
- Complex Configuration Problems
- Supply Chain Management
- Staff Rostering
- Vehicle Routing



Constraint Satisfaction Problem - CSP

- CSP represents a decision optimization problems defining <u>*decision variables*</u> subject to <u>*constraints*</u>
- Typical CSP structure:

1. **Problem Definition (<u>what</u> to do)**
   a. Define Decision Variables with all possible values
   b. Define Constraints on the variables
2. **Problem Resolution (<u>how</u> to do it)**
   a. Find Solution(s) that defines a value for each variable such that all constraints are satisfied or
   b. Find an optimal solution that minimizes/maximizes a certain objective (a function of decision variables)

Simple CSP in Java (JSR-331)

```java
public class Test {

    public static void main(String[] args) {
        // ==== PROBLEM DEFINITION ==================================
        Problem p = ProblemFactory.newProblem("Test");
        // ======= Define variables
        Var x = p.variable("X", 1, 10);
        Var y = p.variable("Y", 1, 10);
        Var z = p.variable("Z", 1, 10);
        Var cost = x.multiply(3).multiply(y).minus(z.multiply(4));
        // ======= Define and post constraints
        p.post(x, "<", y); // X < Y
        p.post(x.plus(y), "=", z); // X + Y = Z

        // === PROBLEM RESOLUTION ====================================
        p.log("=== Find Solution:");
        Solver solver = p.getSolver();
        Solution solution = solver.findSolution();
        if (solution != null)
            solution.log();
        else
            p.log("No Solution");
        p.log("Cost " + cost);
    }
}
```

Solution solution = solver.**findOptimalSolution(cost);**

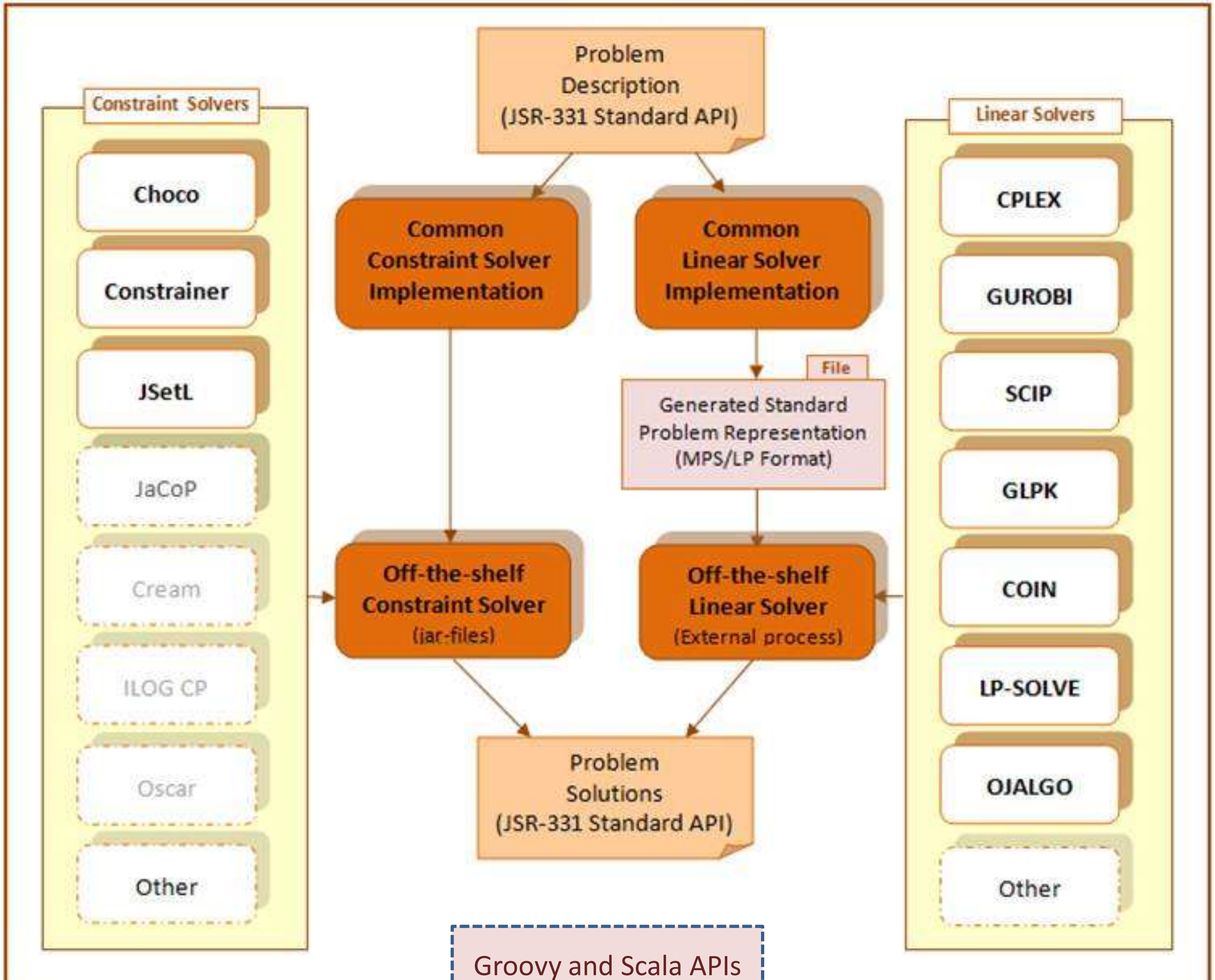
Some Popular Tools

- Java API: JSR-331
  - Choco, Constrainer, JSetL, 7 Linear Solvers
  - Groovy, Scala interfaces
- C++ API
  - IBM/ILOG CP – Commercial (www.ilog.com)
  - Gecode – Open Source (www.gecode.org)
  - New “or-tools” from Google
- CP environments with specialized modeling languages
  - OPL from IBM/ILOG, France (www.ilog.com)
  - MiniZinc from G12 group, Australia (http://www.g12.cs.mu.oz.au)
  - Comet, Brown University (www.comet-online.org)
  - Prolog-based tools (ECLiPSe, SICStus)
  - Drools Planner (Red Hat)
  - 20+ other CP Solvers: http://slash.math.unipd.it/cp/

JSR-331 “Java CP API” Standard

- JSR-331 “Java Constraint Programming API” – an official Java Community Process (JCP) standard www.jcp.org
- JSR-331 covers key concepts and design decisions related to the standard representation and resolution of constraint satisfaction and optimization problems
- Utilizes de-facto standardized decisions from multiple CP solvers
- Integrates CP & LP techniques

JSR-331 Implementations



Groovy and Scala APIs

Use Case "Staff Rostering"

- As the manager, you are required to hire and set the weekly schedule for your employees as follows:
  - Total employees required

Mon	Tue	Wed	Thu	Fri	Sat	Sun
5	8	9	10	16	18	12

  - Available employees:

Employee Type	Total	Cost per Day
F/T	14	$100
P/T	4	$150

- What is the minimal staffing cost?

	M	T	W	T	F	S	S
FT	5	8	9	10	14	14	12
PT	0	0	0	0	2	4	0

Decision “DefineEmployeeSchedule”

- Presented in Excel using OpenRules BDMS
- Utilizes Rule Solver that includes templates for decision, variables, and different constraints

Decision DefineEmployeeSchedule	
Decisions	**Execute Rules**
Define Employee Daily Demand	:= EmployeeDailyDemand()
Define Total Cost	:= DefineTotalCost()

Mon	Tue	Wed	Thu	Fri	Sat	Sun
5	8	9	10	16	18	12

Employee Type	Total	Cost per Day
F/T	14	$100
P/T	4	$150

Decision's Glossary

- Decision Variables:

Glossary glossary			
Decision Variable	**Business Concept**	**Attribute**	**Domain**
Mon FT	Roster	monFT	0-14
Mon PT		monPT	0-4
Tue FT		tueFT	0-14
Tue PT		tuePT	0-4
Wed FT		wedFT	0-14
Wed PT		wedPT	0-4
Thu FT		thuFT	0-14
Thu PT		thuPT	0-4
Fri FT		friFT	0-14
Fri PT		friPT	0-4
Sat FT		satFT	0-14
Sat PT		satPT	0-4
Sun FT		sunFT	0-14
Sun PT		sunPT	0-4
Total Cost		totalCost	0-20000

Decision Tables

DecisionTable EmployeeDailyDemand				
ActionXoperYcompareZ				
Variable	Arith Oper	Variable	Compare Oper	Value
Mon FT	+	Mon PT	=	5
Tue FT	+	Tue PT	=	8
Wed FT	+	Wed PT	=	9
Thu FT	+	Thu PT	=	10
Fri FT	+	Fri PT	=	16
Sat FT	+	Sat PT	=	18
Sun FT	+	Sun PT	=	12

Mon	Tue	Wed	Thu	Fri	Sat	Sun
5	8	9	10	16	18	12

Employee Type	Total	Cost per Day
F/T	14	$100
P/T	4	$150

DecisionTable DefineTotalCost		
ActionScalProd		
Name of the Scalar Product	**Numbers**	**Variables**
Total Cost	100,150,100,150,100,150,100,150,100,150,100,150,100,150	Mon FT, Mon PT, Tue FT, Tue PT, Wed FT, Wed PT, Thu FT, Thu PT, Fri FT, Fri PT, Sat FT, Sat PT, Sun FT, Sun PT

Run Decision from Java

```java
import com.openrules.ruleengine.Decision;

public class Main {

    public static void main(String[] args) {

        String fileName = "file:rules/Decision.xls";
        System.setProperty("OPENRULES_MODE", "Solve");
        Decision decision = new Decision("DefineEmployeeSchedule",fileName);
        decision.put("MaxSolutions", "30");
        decision.put("Minimize","Total Cost");
        decision.execute();
        decision.execute("PrintSolution");
    }
}
```

Decision Results

…
Found a solution with Total Cost[8700]
Found a solution with Total Cost[8650]
Found a solution with Total Cost[8600]
Found a solution with Total Cost[8550]
Found a solution with Total Cost[8500]
Found a solution with Total Cost[8450]
Found a solution with Total Cost[8400]
Found a solution with Total Cost[8350]
Found a solution with Total Cost[8300]
Found a solution with Total Cost[8250]
Found a solution with Total Cost[8200]
Found a solution with Total Cost[8150]
Found a solution with Total Cost[8100]
Found a solution with Total Cost[8100]
*** Execution Profile ***
Number of Choice Points: 94360
Number of Failures: 94333
Occupied memory: 93172496
Execution time: 14885 msec

==== Optimal Solution =====

	M	T	W	T	F	S	S
FT	5	8	9	10	14	14	12
PT	0	0	0	0	2	4	0

Total Cost: 8100
=========================

The Same Decision Model in Java

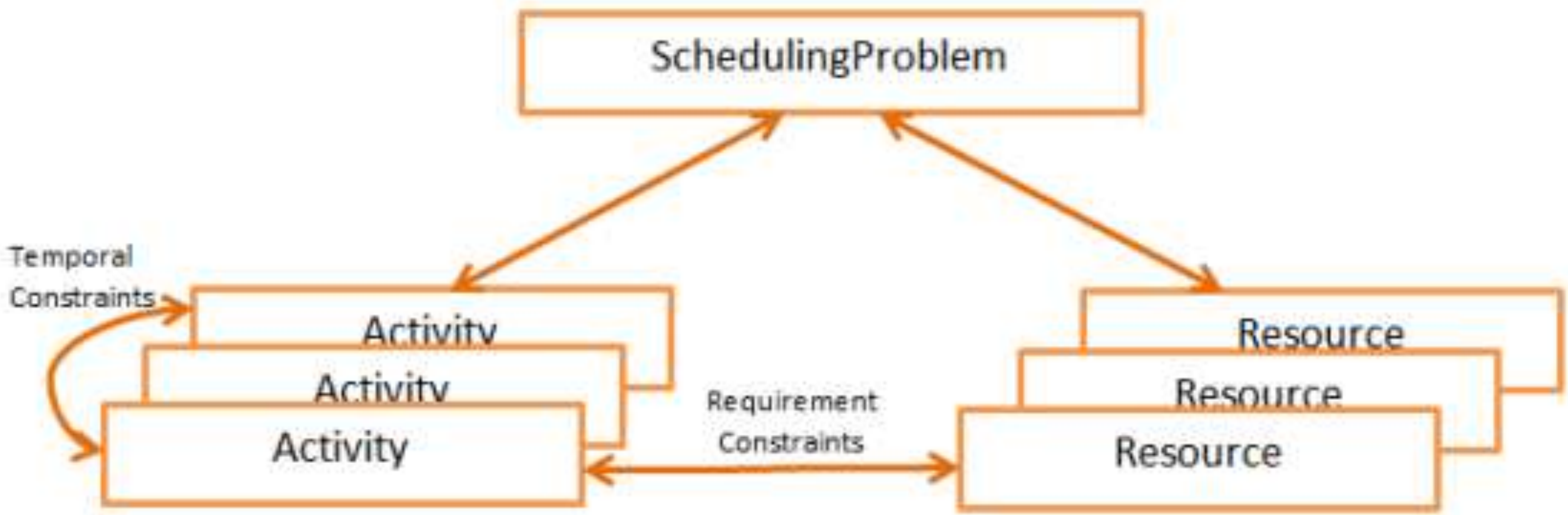
(JSR-331)

```java
public static void main(String[] args) {
    Problem p = ProblemFactory.newProblem("EmployeeRostering1");
    // Define FT and PT variables
    int maxFT = 14;
    int maxPT = 4;
    Var monFT = p.variable("MonFT", 0, maxFT);
    Var monPT = p.variable("MonPT", 0, maxPT);
    Var tueFT = p.variable("TueFT", 0, maxFT);
    Var tuePT = p.variable("TuePT", 0,maxPT);
    Var wedFT = p.variable("WedFT", 0, maxFT);
    Var wedPT = p.variable("WedPT", 0, maxPT);
    Var thuFT = p.variable("ThuFT", 0, maxFT);
    Var thuPT = p.variable("ThuPT", 0, maxPT);
    Var friFT = p.variable("FriFT", 0, maxFT);
    Var friPT = p.variable("FriPT", 0, maxPT);
    Var satFT = p.variable("SatFT", 0, maxFT);
    Var satPT = p.variable("SatPT", 0, maxPT);
    Var sunFT = p.variable("SunFT", 0, maxFT);
    Var sunPT = p.variable("SunPT", 0, maxPT);

    // Post daily constraints
    p.post(monFT.plus(monPT),"=",5);
    p.post(tueFT.plus(tuePT),"=",8);
    p.post(wedFT.plus(wedPT),"=",9);
    p.post(thuFT.plus(thuPT),"=",10);
    p.post(friFT.plus(friPT),"=",16);
    p.post(satFT.plus(satPT),"=",18);
    p.post(sunFT.plus(sunPT),"=",12);

    // Define costs
    int[] costs = {100,150,100,150,100,150,100,150,100,150,100,150,100,150};
    Var[] vars = {monFT,monPT,tueFT,tuePT,wedFT,wedPT,thuFT,thuPT,friFT,friPT,satFT,satPT,sunFT,sunPT};
    Var totalCost = p.scalProd(costs, vars);
    p.add("TotalCost",totalCost);
```

Decision Modeling Considerations

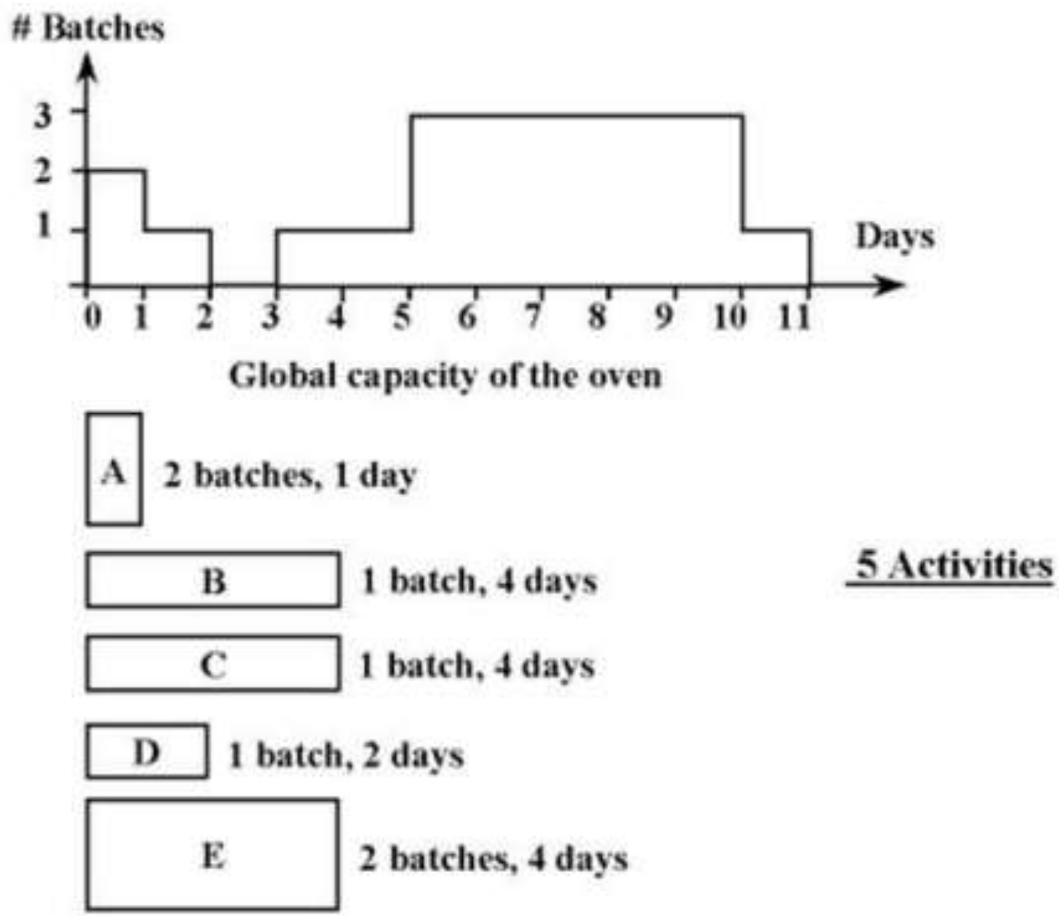
- Choice of decision variable should:
  - Allow to express problem constraints
  - Lead to an efficient decision execution avoiding a "combinatorial explosion"
- Example. Let's add constraints:
  - each F/T employee should work five days in a row
  - each P/T employee should work two days in a row
  - ***The old choice of decision variables would not work anymore - we have to change the model!***

JSR-331 Scheduler



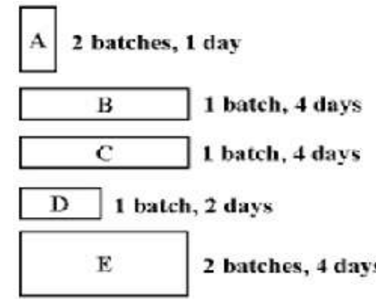
Resource Allocation Problem

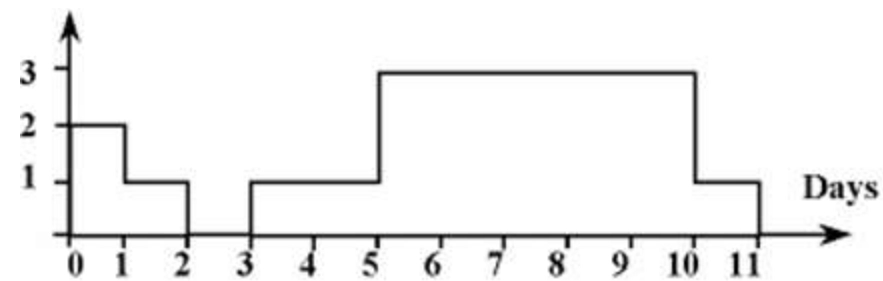
The following problem deals with activities that require a common resource. Let's consider 5 different orders (activities) that fire batches of bricks in an oven (a resource with a limited capacity). Each order 's size and duration, as well as the oven's capacity, are described in the following figure:



Sample Problem Implementation
(Java with JSR-331 Scheduler)

```
Schedule schedule = ScheduleFactory.newSchedule(“oven”0, 11);
Activity A = schedule.addActivity(1, "A");
Activity B = schedule.addActivity(4, "B");
Activity C = schedule.addActivity(4, "C");
Activity D = schedule.addActivity(2, "D");
Activity E = schedule.addActivity(4, "E");
Resource oven = schedule.addResource(3, "oven");
oven.setCapacityMax(0, 2);
oven.setCapacityMax(1, 1);
oven.setCapacityMax(2, 0);
oven.setCapacityMax(3, 1);
oven.setCapacityMax(4, 1);
oven.setCapacityMax(10, 1);
// Resource Constraints
A.requires(oven, 2).post();
B.requires(oven, 1).post();
C.requires(oven, 1).post();
D.requires(oven, 1).post();
E.requires(oven, 2).post();
// Find Solution
schedule.scheduleActivities();
schedule.displayActivities();
```

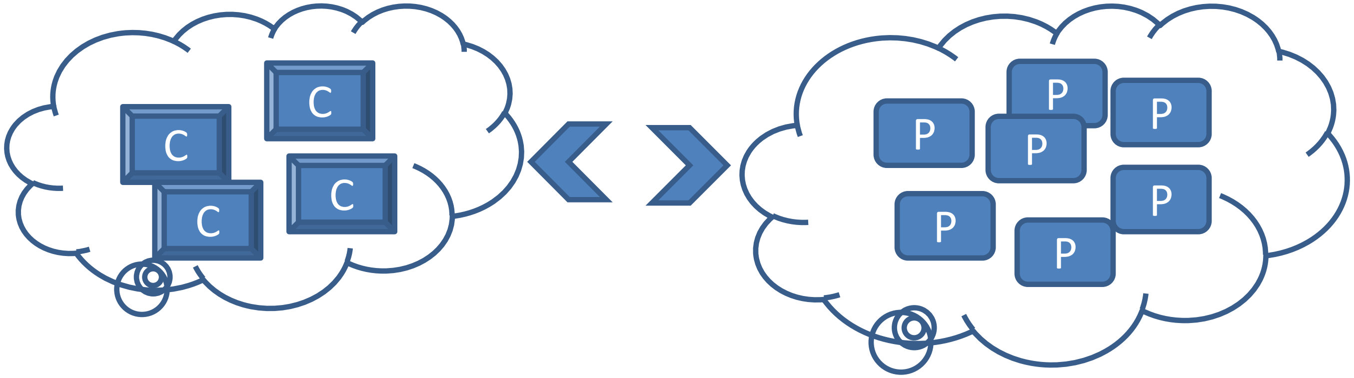




SOLUTION:
A[5 -- 1 --> 6) requires oven[2]
B[3 -- 4 --> 7) requires oven[1]
C[7 -- 4 --> 11) requires oven[1]
D[0 -- 2 --> 2) requires oven[1]
E[6 -- 4 --> 10) requires oven[2]

Use Case “Cloud Balancing”

- You have a number of cloud computers and need to run a number of processes on those computers.
- Each process requires certain CPU power, RAM, and network bandwidth and incurs a certain maintenance cost (which is fixed per computer)
- Objective: assign process to computers while minimize the total maintenance cost.

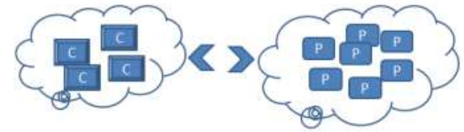


Variable Computer

- Input classes: CloudComputer, CloudProcess
- Decision variables are in this class:

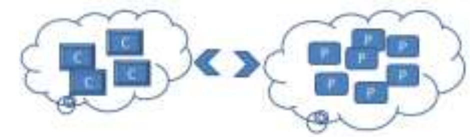
```
public class VarComputer {
    CloudComputer computer;
    Var[] processVars; // processVars[i] = 1 means this computer is used by the i-th process

    public VarComputer(Problem p, CloudComputer computer, CloudProcess[] processes,
            int[] requiredMemories, int[] requiredCpuPowers, int[] requiredNetworkBandwidths) {
        this.computer = computer;
        processVars = new Var[processes.length];
        for (int i = 0; i < processes.length; i++) {
            String name = "P" + processes[i].getId() + "C" + computer.getId();
            processVars[i] = p.variable(name,0,1);
        }
        p.post(requiredMemories, processVars,"<=",computer.getMemory());
        p.post(requiredCpuPowers, processVars,"<=",computer.getCpuPower());
        p.post(requiredNetworkBandwidths, processVars,"<=",computer.getNetworkBandwidth());
    }

    public Var[] getProcessVars() {
        return processVars;
    }
}
```

Modeling and Search for an Optimal Solution

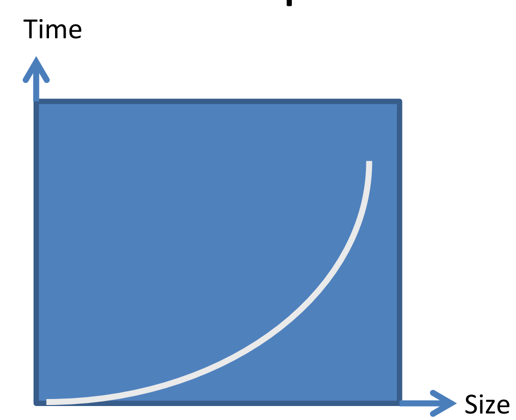


- **A small problem “4 x12”** with a constraint solver
  - 4 computers and 12 processes
  - “Brute force” approach: 650mills
  - “Sort processes first” approach: 490 mills
- **A larger problem “10 x 20”**
  – Constraint solver takes 30 seconds
    - (50x longer) and only when we set a time limit
  – Linear Solver (identical source code, just different jars in classpath)
    - Finds an optimal solution in 1,200 milliseconds

Modeling and Search for an Optimal Solution (2)



- **A large problem “50 x100”**
  - 50 computers and 100 processes
  - Constraint solver requires special selectors and time limits
  - Linear Solver takes 2.5 hours to find an optimal solution
- **A huge problem “5,000 x 55,000”**
  - Offered at the recent ROADEF-2012 competition
  - The winners found the best solution within 5 mins using a unique selection of subsets of processes and computers and a specially written solver



Smarter Decision Modeling (1)

- **Minimizing Combinatorial Complexity**
  - "Combinatorial complexity" is the number of *leaves* on the search tree
  - **Initial Decision Model**
    - Assigning P processes to C computers
    - P decision variables with domain [0;1] for each computer
  - **Alternative Decision Model**
    - Assigning C computers to P processes
    - 1 decision variable with domain [1;C] for each process
  - **Compare combinatorial complexity**:
    - Roughly: compare C^P vs. P^C
    - E.g. $5^3 = 125$ while $3^5 = 243$

Smarter Decision Modeling (2)

- **Avoiding Symmetry**
  - Does not make sense to alternate (permute) between computers/processes with identical characteristics
  - Grouping by Type
    - Create groups of computers with identical resource characteristics
    - Create groups of processes with identical resource requirements
  - Introducing an order among decision variables with identical characteristics
  - Using Set Constrained Variables

Smarter Search Strategies (1)

- Adding time limits for:
  - Search of one solution
  - The overall search
- CP solvers provide many search strategies for selecting variables and values to try first, e.g.
  - Famous n-Queens problem: using a selector MIN_DOMAIN_MIN_VALUE improves performance 1,000 times
  - Drools Planner successfully addressed large process-computer assignment problems for 2,500 processes by using special “just-in-time” selectors
- CP/LP tools provide different optimization options that may be tried without changing a decision model

Smarter Search Strategies (2)

- There is no single rule of thumb for discovering a good strategy. One may try strategies that:
  - are deduced from common sense
  - use “know-how” from the problem domain
  - borrow proven methods from Operation Research (OR)
- However, larger problems may still require specialized decision models and even specialized solvers
  - Recent ROADEF-2012 competition offered a problem with up to 55,000 processes and 5,000 computers
  - Find the best possible solution within 5 minutes
  - The winners used a special selection of subsets of processes and computers on each iteration and a specially written solver (utilizing a “large neighborhood search”)

Role of Experts

- BR+CP combination creates a powerful while intuitive decision modeling and optimization framework!
- However, its simplicity should not mislead business analysts (or even Java developers) that now they can solve any decision optimization problems
- No API can replace an expert when you deal with large size problems

Conclusion

- Many practical Decision Optimization problems may be successfully modeled and solved by subject matter experts using off-the-shelf CP/LP tools such as Rule Solver
- The JSR-331 standard gives all BR vendors an opportunity to add a true optimization component to their product offerings
- The best results are achieved when a subject matter expert works together with an OR specialist

Q&A

www.OpenRules.com