Introduction to Google OR-Tools

Google OR-Tools

PROS Lab Members

Flavio Corradini

FULL PROFESSOR

Andrea Polini

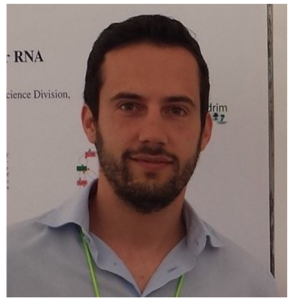
ASSOCIATE PROFESSOR

Barbara Re

ASSOCIATE PROFESSOR

Francesco Tiezzi

ASSOCIATE PROFESSOR



Andrea Morichetta

RESEARCH FELLOW

Fabrizio Fornari

POSTDOCTORAL RESEARCHER

Lorenzo Rossi

POSTDOCTORAL RESEARCHER

Marco Piangerelli

POSTDOCTORAL RESEARCHER

Alessandro Marcelletti

PHD STUDENT

Caterina Luciani

PHD STUDENT

Khalid Bourr

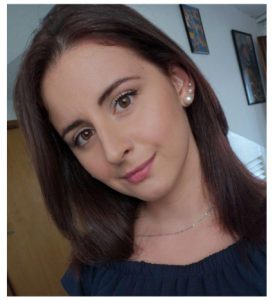
PHD STUDENT

Ivan Compagnucci

PHD STUDENT

Sara Pettinari

PHD STUDENT



Arianna Fedeli

PHD STUDENT

and…

- Morena Barboni
- Vincenzo Nucci
- Ahmad Ronaghikhameneh
- Umair Qureshi

Ivan Compagnucci

- Ph.D. student at UNICAM
- PROS Lab Member

PROS Lab
pros.unicam.it

PROS
PROcesses and Services Lab

pros@unicam.it

http://pros.unicam.it

Interests

- Business Process Management
- BPMN
- **BP** & **IoT** modeling and enactment

Ivan Compagnucci

PHD STUDENT

ivan.compagnucci@unicam.it

https://www.linkedin.com/in/ivan-compagnucci/

ivan.compagnucci@studenti.unicam.it

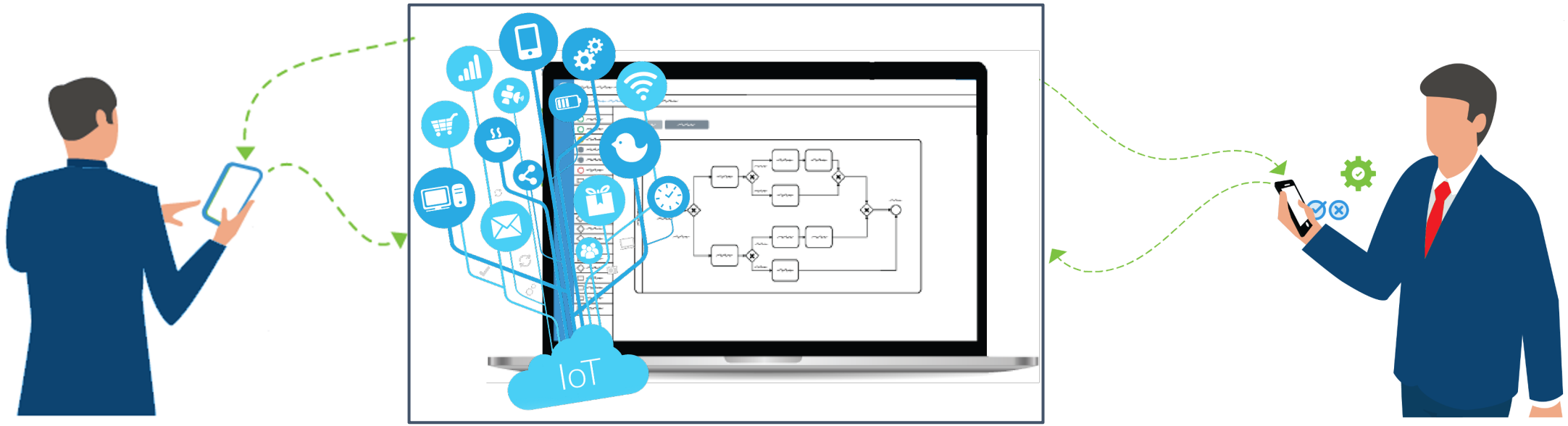
Research Topics

Internet of Things

Network of interconnected devices that collect and exchange data to monitor, control or transfer relevant information so as to be able to perform consequent intelligent actions

Business Process

A set of activities, tasks or actions to carry out a specific organizational goal such as a service or a product



Business Process Meet Internet of Things

- **Design** and **monitoring** of the smart environment for a **better execution**, **safety** and **less complexity**
- **Bridging the gap** between the high level of the Business Process and the low level of the IoT technologies
- Programming of "**dependencies between independent devices**“ in a process-oriented vision

Process-Oriented Modelling Notations for Internet of Things

Ivan Compagnucci
ivan.compagnucci@unicam.it
Computer Science Division, Science and Technology School, University of Camerino

PROS Lab
pros.unicam.it

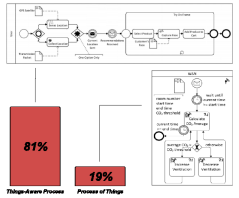
Business Process meet Internet of Things

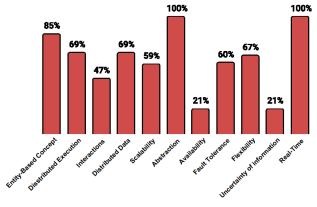
The **Internet of Things** term refers to the inter-networking of physical objects embedded with electronics hardware, software, sensors, actuators, and network connectivity.

The **Business Process Management** is a well-established discipline that deals with the analysis, design, implementation, execution, monitoring, and evolution of business processes.

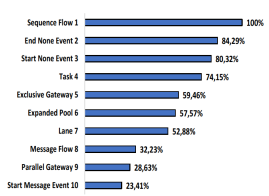
SLR Research Questions

- RQ1. Which are the **relevant modelling perspectives** to consider when modelling IoT-Aware business processes?
- RQ2. What are the **IoT requirements supported** by notations used to model IoT-Aware business processes?
- RQ3. Which are the **modelling notations proposed** and adopted to model IoT-Aware business processes?

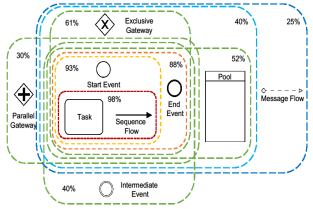




Trends on the Usage of BPMN 2.0 Standard Notation



Frequency Distribution of BPMN Elements

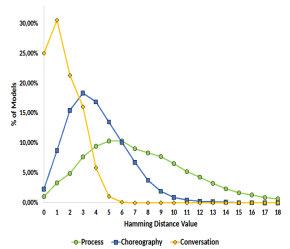


Subsets of common BPMN Elements

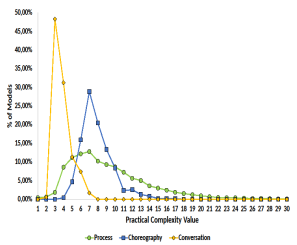
There isn't a standard *de facto* for modeling smart scenarios in a process-oriented way. Existing solutions **mainly use BPMN** or **an extension of it**.

14% Not BPMN
86% BPMN or BPMN Extension

However, there is no a well-established and mature approach that **fully meet the requirements to represent an IoT scenario**.



Hamming Distance of BPMN Elements



Practical Complexity of BPMN Elements

Element One	Element Two	ρ
Process		
Receive Task	Send Task	0.76
Sequence Flow	Task	0.75
Sequence Flow	Exclusive Gateway	0.73
Intermediate Throw Link Event	Intermediate Catch Link Event	0.72
End Event	Start Event	0.68
Start Event	Expanded Sub-Process	0.64
Intermediate Throw Message Event	Intermediate Catch Message Event	0.59
Association Data Output	Data Object	0.59
Expanded Pool	Lane	0.58
Sequence Flow	End Event	0.55
Exclusive Gateway	Task	0.53
Message Flow	Intermediate Catch Message Event	0.53
Expanded Pool	Message Flow	0.52
End Event	Expanded Sub-Process	0.50
Intermediate Throw Signal Event	Intermediate Catch Signal Event	0.50
Choreography		
Choreography Participant	Choreography Task	0.96
Choreography Task	Choreography Message	0.54
Choreography Participant	Choreography Message	0.51
Conversation		
Conversation Link	Conversation	0.98
Conversation Link	Collapsed Pool	0.54
Conversation	Collapsed Pool	0.54

BPMN Elements Pairs Correlation

References

[1] I. Compagnucci, F. Corradini, F. Fornari, A. Polini, B. Re and F. Tiezzi. Modelling Notations for IoT-Aware Business Processes: A Systematic Literature Review. BPM Workshop, Vol. 397, 108-121, 2020.
[2] I. Compagnucci, F. Corradini, F. Fornari, and B. Re. Trends on the Usage of BPMN 2.0 from Publicly Available Repositories. In: Perspectives in Business Informatics Research, Vol. 430, 84-99, 2021.

Findings

There isn't a standard de facto for modeling smart scenarios in a process- oriented way. Existing solutions (86%) **mainly use BPMN or an extension of it**

However, **there is no a well-established and mature approach** that fully meet the requirements to represent an IoT scenario.

...Constraint programming can improve IoT systems?

Constraint Programming

Constraint programming is a powerful paradigm for **solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence (AI), operations research, algorithms, and graph theory.**

The basic idea in constraint programming is that **the user states the constraints,** and a general-purpose **"constraint solver" is used to solve them.**

Constraint Programming: Everyday-life example

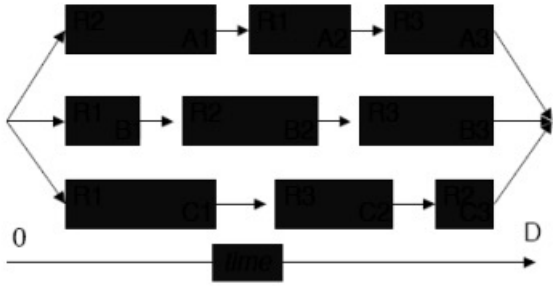
Sudoku

- Rules for inserting numbers in the table

			9		2			
	4						5	
		2				3		
2								7
			4	5	6			
6								9
		7				8		
	3						4	
			2		7			

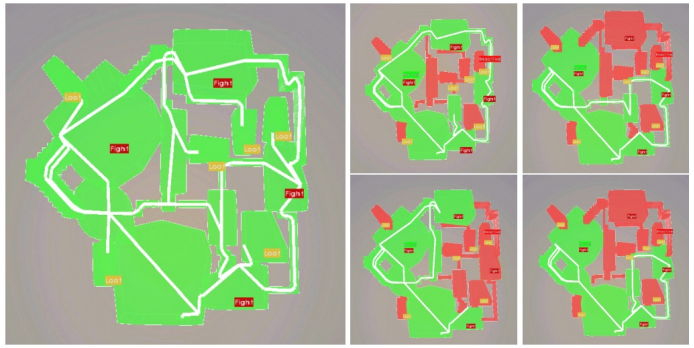
CPU's Job Scheduling

- Constraints on determing which process should be executed



Gaming: Procedural Dungeon Simulation

- Procedural dungeon generation
  in an open world/universe context



What is OR Tools?

OR-Tools is an open source software suite for **optimization, for solve problems in vehicle routing, network flows, integer and linear programming, and constraint programming.**

Google OR-Tools

What is OR Tools? - Examples

- **Vehicle routing:** Find optimal routes for vehicle fleets that pick up and deliver packages given constraints (e.g., "this truck can't hold more than 20,000 pounds" or "all deliveries must be made within a two-hour window").
- **Scheduling:** Find the optimal schedule for a complex set of tasks, some of which need to be performed before others, on a fixed set of machines, or other resources.
- **Bin packing:** Pack as many objects of various sizes as possible into a fixed number of bins with maximum capacities.

*In most cases, problems like these have **a vast number of possible solutions**. OR-Tools uses state-of-the-art algorithms to narrow down the search set**, in order to find an optimal (or close to optimal) solution.***

OR Tools: Identify the type of Solvers

There are **many different types of optimization problems in the world**. For each type of problem, there are **different approaches and algorithms for finding an optimal solution**. Before you can start writing a program to solve an optimization problem, you need to **identify what type of problem you are dealing with, and then choose an appropriate solver** — an algorithm for finding an optimal solution.

These are the types of problems that OR-Tools solves:

- Linear optimization
- Mixed-Integer optimization
- Constraint optimization
- Network flows/ Routing
- Assignment
- Scheduling

OR Tools: Solvers

OR-Tools include **solvers** for:

- **Constraint Programming: A set of techniques for finding feasible solutions to a problem expressed as *constraints*** (e.g., a room can't be used for two events simultaneously, or the distance to the crops must be less than the length of the hose, or no more than five TV shows can be recorded at once).
- **Linear and Mixed-Integer Programming:** The Glop linear optimizer **finds the optimal value of a linear objective function**, given a set of linear inequalities as constraints (e.g., assigning people to jobs, or finding the best allocation of a set of resources while minimizing cost
- **Scheduling: *Scheduling problems* involve assigning resources to perform a set of tasks at specific times**. An important example is the ***job shop problem***, in which multiple jobs are processed on several machines. Each job consists of a sequence of tasks, which must be performed in a given order, and each task must be processed on a specific machine. The problem is to assign a schedule so that all jobs are completed in as short an interval of time as possible.

OR Tools: Solvers

OR-Tools include **solvers** for:

- **Network Flow:** Many optimization problems **can be represented by a directed graph** consisting of **nodes** and directed **arcs** between them. For example**, transportation problems**, in which goods are shipped across a railway network, can be represented by a graph in which the arcs are rail lines and the nodes are distribution centers. **The problem is to assign the amount of goods to be shipped across each arc** so that the total quantity being transported is as large as possible.
- **Assignment:** *Assignment* problems involve **assigning a group of agents** (say, workers or machines) **to a set of tasks, where there is a fixed cost for assigning each agent to a specific task**. The problem is to find the assignment with the least total cost.

OR-Tools Awards

OR-Tools won four gold medals in the 2021 MiniZinc Challenge, the *International Constraint Programming Competition*

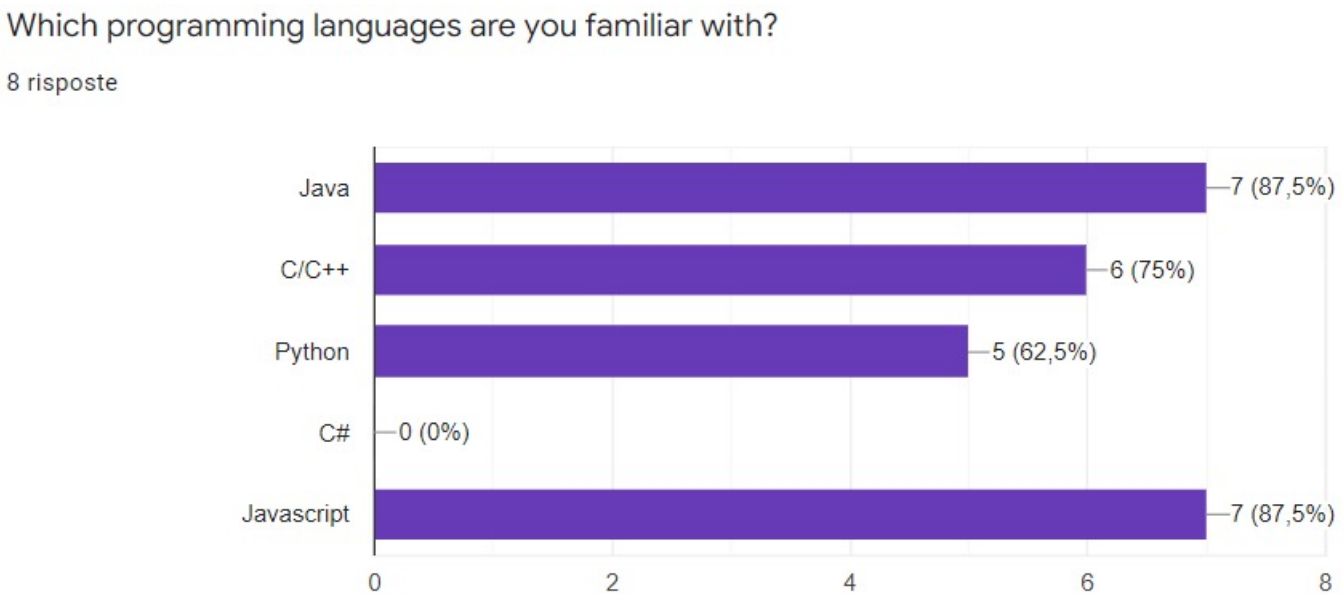
[1]

Category	Gold	Silver	Bronze
Fixed	OR-Tools	JaCoP	SICStus Prolog
Free	OR-Tools	PicatSAT	iZplus
Parallel	OR-Tools	PicatSAT	iZplus/Choco 4
Open	OR-Tools	PicatSAT	iZplus/Choco 4
Local Search	Yuck	OscaR/CBLS	

[1] https://www.minizinc.org/challenge2021/challenge.html

OR Tools in practice

OR-Tools is written in C++, but **you can also use it with Python, Java, or C#.**



Considering your skill, we will use OR-Tools in **Java**

OR Tools: Installation

Download the following resources:

- **“Constraints”:**
  - You must have the **Microsoft Visual C++ Redistributable for Visual Studio 2019.** https://visualstudio.microsoft.com/downloads/?q=Visual+C%2B%2B+Redistributable+for+Visual+Studio
  - You must also have a **Java JDK 64 bit, version 8.0 or later installed**. https://java.com/en/download/help/download_options.html
  - You must also have a **Maven 64 bit installed**. https://maven.apache.org/download.cgi

- **Visual Studio Code 2022:** As IDE.
- **OR-Tools Binary Distribution:**
  1. Visit: **https://developers.google.com/optimization/install**
  2. Select the distribution depending on your Operating System.
  3. Unzip

OR Tools: Installation

Testing the installation (Inside the unzipped folder):

```
> tools\make test_java
```

You should be able to have this output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] --- maven-install-plugin:3.0.0-M1:install-file (default-cli) @ standalone-pom ---
[INFO] Installing C:\OR-Tools\ortools-win32-x86-64-9.3.10497.jar to C:\Users\User\.m2\rep
ols-win32-x86-64\9.3.10497\ortools-win32-x86-64-9.3.10497.jar
[INFO] Installing C:\Users\User\AppData\Local\Temp\ortools-win32-x86-64-9.3.1049748104416
r\.m2\repository\com\google\ortools\ortools-win32-x86-64\9.3.10497\ortools-win32-x86-64-9
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.864 s
[INFO] Finished at: 2022-04-14T16:07:39+02:00
[INFO] ------------------------------------------------------------------------
```

Possible errors

Make sure you:

- **Defined environmental variables (PATH):**
  - **For JDK, Maven, cmd view, …**
- Microsoft Visual C++ Redistributable **is up to date!**
- Install MinGW (If "*make*" command doesn't work)
- …

OR Tools: What is an optimization problem?

The goal of **optimization** is to find the *best* solution to a problem out of a large set of possible solutions. (Sometimes you'll be satisfied with finding any feasible solution; OR-Tools can do that as well.)

A typical optimization problem:

Here's a typical optimization problem. Suppose that a **shipping company delivers packages to its customers using a fleet of trucks**. Every day, the company must assign packages to trucks, and then choose a route for each truck to deliver its packages. Each possible assignment of packages and routes has a cost, based on the total travel distance for the trucks, and possibly other factors as well. The problem is to choose the assignments of packages and routes that has the least cost.

OR Tools: What is an optimization problem?

Like all optimization problems, this problem has the following 2 elements:

- **The Objective:** The quantity you want to optimize. Here, the objective is to **minimize cost**. To set up an optimization problem, you need to **define a function that calculates the value of the objective for any possible solution**. This is called the ***objective function***. **In the preceding example, the objective function would calculate the total cost of any assignment of packages and routes**.

*An **optimal solution** is one for which the value of the objective function is the best.*

- **The Constraints:** Restrictions on the **set of possible solutions, based on the specific requirements of the problem**. For example, if the shipping company can't assign packages above a given weight to trucks, this would impose a constraint on the solutions.

*A **feasible solution** is one that satisfies all the given constraints for the problem, without necessarily being optimal.*

Solving a linear optimization problem in Java

A linear optimization example:

- **Maximize** $3x + y$ subject to the following constraints:
- Both the objective function and the constraints are given by linear expressions, which makes this a linear problem.

$$
\begin{array}{lllll}
0 & \leq & x & \leq & 1 \\
0 & \leq & y & \leq & 2 \\
 & & x+y & \leq & 2
\end{array}
$$

Main 5 steps to solve the problem with OR-Tools:

1. Import the required libraries;
2. Declare the **Solver**;
3. Create variables;
4. Define the constraints;
5. Define the objective function;
6. Invoke the Solver and display results.

Solving a linear optimization problem in Java

1. Import the required libraries:

```
import com.google.ortools.Loader;
import com.google.ortools.linearsolver.MPConstraint;
import com.google.ortools.linearsolver.MPObjective;
import com.google.ortools.linearsolver.MPSolver;
import com.google.ortools.linearsolver.MPVariable;
```

2. Declare the Solver:

```
// Create the linear solver with the GLOP backend.
MPSolver solver = MPSolver.createSolver("GLOP");
```

MPSolver is a wrapper for solving any linear programming or mixed integer programming problems.

3. Create Variables

```
// Create the variables x and y.
MPVariable x = solver.makeNumVar(0.0, 1.0, "x");
MPVariable y = solver.makeNumVar(0.0, 2.0, "y");

System.out.println("Number of variables = " + solver.numVariables());
```

Solving a linear optimization problem in Java

4. Define the Constraints

```
// Create a linear constraint, 0 <= x + y <= 2.
MPConstraint ct = solver.makeConstraint(0.0, 2.0, "ct");
ct.setCoefficient(x, 1);
ct.setCoefficient(y, 1);

System.out.println("Number of constraints = " + solver.numConstraints());
```

5. Define the Objective of the function

```
// Create the objective function, 3 * x + y.
MPObjective objective = solver.objective();
objective.setCoefficient(x, 3);
objective.setCoefficient(y, 1);
objective.setMaximization();
```

6. Invoke the solver and display the results

```
solver.solve();
System.out.println("Solution:");
System.out.println("Objective value = " + objective.value());
System.out.println("x = " + x.solutionValue());
System.out.println("y = " + y.solutionValue());
```

The first two constraints:

- $0 \leq x \leq 1$;
- $0 \leq y \leq 2$.

Are already set by the definitions of the variables. The following code defines the constraint:

- $x + y \leq 2$.

The method **setCoefficient** sets the coefficients of x and y in the expression for the constraint.

The method **setMaximization** declares this to be a maximization problem

Solving a linear optimization problem in Java

You can run the program as follow:

1. **Open a command window at the top level of the directory where you installed OR-Tools, and enter:**

```
$ make build SOURCE=relative/path/to/my_program.java
```

2. **Then run the program:**

```
$ make run SOURCE=relative/path/to/my_program.java
```

3. **Display results:**

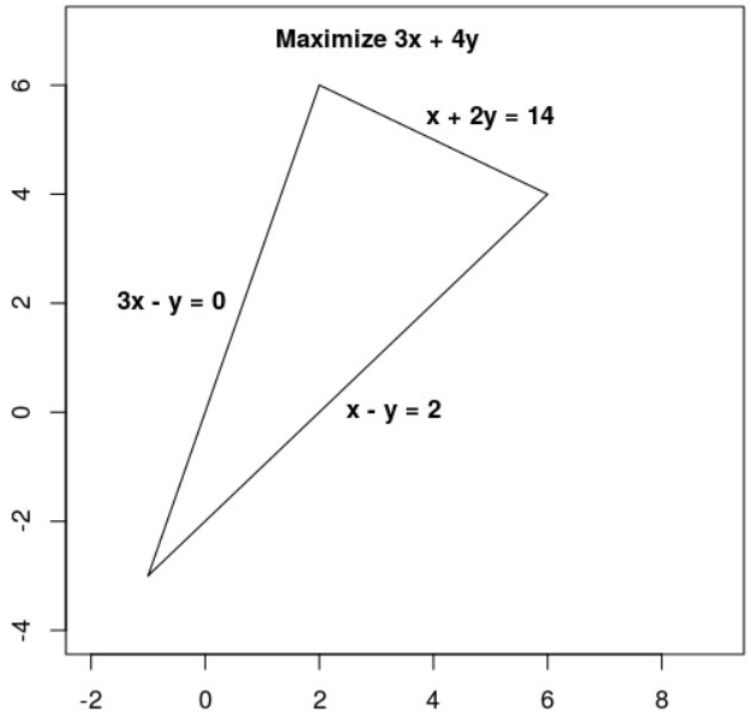
```
Number of variables = 2
Number of constraints = 1
Solution:
Objective value = 4.0
x = 1.0
y = 1.0
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.619 s
[INFO] Finished at: 2022-04-17T17:31:04+02:00
[INFO] ------------------------------------------------------------------------
```

Another linear optimization problem in Java

A linear optimization example:

- **Maximize** $3x + 4y$ subject to the following constraints:
- Both the objective function and the constraints are given by linear expressions, which makes this a linear problem.

$x + 2y$	$\leq$	14
$3x - y$	$\geq$	0
$x - y$	$\leq$	2



The constraints define the feasible region, which is the triangle shown below, including its interior.

Another linear optimization problem in Java

Main 6 steps to solve the problem with OR-Tools:

1. Import the required libraries;
2. Declare the **Solver**;
3. Define variables;
4. Define the constraints;
5. Define the objective function;
6. Invoke the Solver and display results.

Another linear optimization problem in Java

1. **Import the required libraries:**

```
import com.google.ortools.Loader;
import com.google.ortools.linearsolver.MPConstraint;
import com.google.ortools.linearsolver.MPObjective;
import com.google.ortools.linearsolver.MPSolver;
import com.google.ortools.linearsolver.MPVariable;
```

2. **Declare the Solver:**

```
// Create the linear solver with the GLOP backend.
MPSolver solver = MPSolver.createSolver("GLOP");
```

3. **Create Variables**

```
double infinity = java.lang.Double.POSITIVE_INFINITY;
// x and y are continuous non-negative variables.
MPVariable x = solver.makeNumVar(0.0, infinity, "x");
MPVariable y = solver.makeNumVar(0.0, infinity, "y");
System.out.println("Number of variables = " + solver.numVariables());
```

Variables *x* and *y* whose **values are in the range from 0 to infinity.**

Another linear optimization problem in Java

4. Define the Constraints

```
// x + 2*y <= 14.
MPConstraint c0 = solver.makeConstraint(-infinity, 14.0, "c0");
c0.setCoefficient(x, 1);
c0.setCoefficient(y, 2);

// 3*x - y >= 0.
MPConstraint c1 = solver.makeConstraint(0.0, infinity, "c1");
c1.setCoefficient(x, 3);
c1.setCoefficient(y, -1);

// x - y <= 2.
MPConstraint c2 = solver.makeConstraint(-infinity, 2.0, "c2");
c2.setCoefficient(x, 1);
c2.setCoefficient(y, -1);
System.out.println("Number of constraints = " + solver.numConstraints());
```

Constraints:

- ***C0***: x+2y <= 14;
- ***C1***: 3x-y >= 0.
- ***C2***: x-y <= 2.

5. Define the Objective of the function

```
// Maximize 3 * x + 4 * y.
MPObjective objective = solver.objective();
objective.setCoefficient(x, 3);
objective.setCoefficient(y, 4);
objective.setMaximization();
```

The following code defines the objective function, 3*x* + 4*y*, and specifies that this is a **maximization** problem

Solving a linear optimization problem in Java

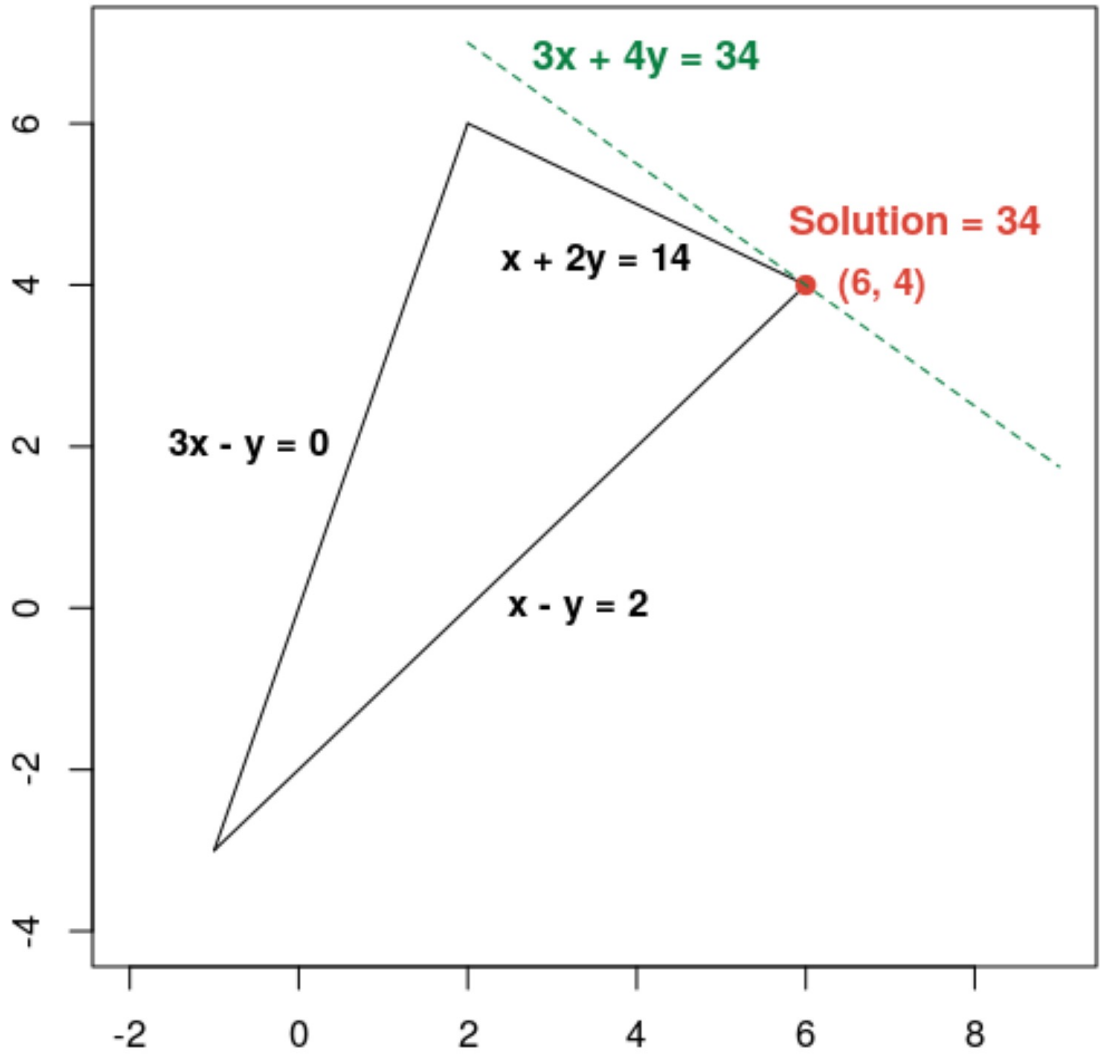
4. Invoke the solver

```
final MPSolver.ResultStatus resultStatus = solver.solve();
```

5. Display results

```
if (resultStatus == MPSolver.ResultStatus.OPTIMAL) {
  System.out.println("Solution:");
  System.out.println("Objective value = " + objective.value());
  System.out.println("x = " + x.solutionValue());
  System.out.println("y = " + y.solutionValue());
} else {
  System.err.println("The problem does not have an optimal solution!");
}
```

Solving a linear optimization problem in Java



The dashed green line is defined by setting the objective function equal to its optimal value of 34.

Any line whose equation has the form $3x + 4y = c$ is parallel to the dashed line, and 34 is the largest value of c for which the line intersects the feasible region

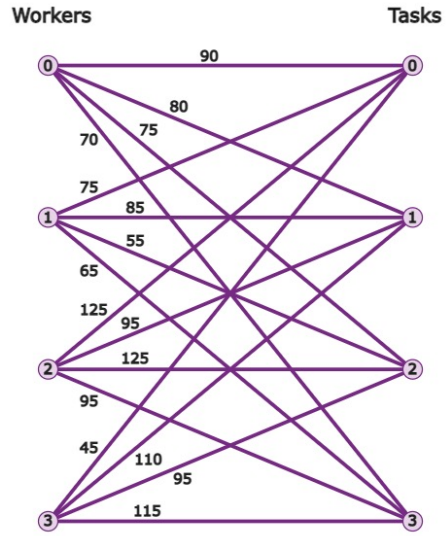
```
Number of variables = 2
Number of constraints = 3
Solution:
x = 6.0
y = 4.0
Optimal objective value = 34.0
```

Mixed-Integer optimization problem in Java

Linear optimization problems that require some of the variables to be integers are called *Mixed Integer Programs* (MIPs). These variables can arise in a couple of ways:

- **Integer Variables**: **that represent numbers of items, such as cars or television sets, and the problem is to decide how many of each item to manufacture in order to maximize profit**. Typically, such problems can be set up as standard linear optimization problems, with the added requirement that the variables must be integers.

- **Boolean variables: that represent decisions with 0-1 values.** As an example, consider a problem that involves assigning workers to tasks. To set up this type of problem, you can define Boolean variables $x_{i,j}$ that equal 1 if worker i is assigned to task j, and 0 otherwise.

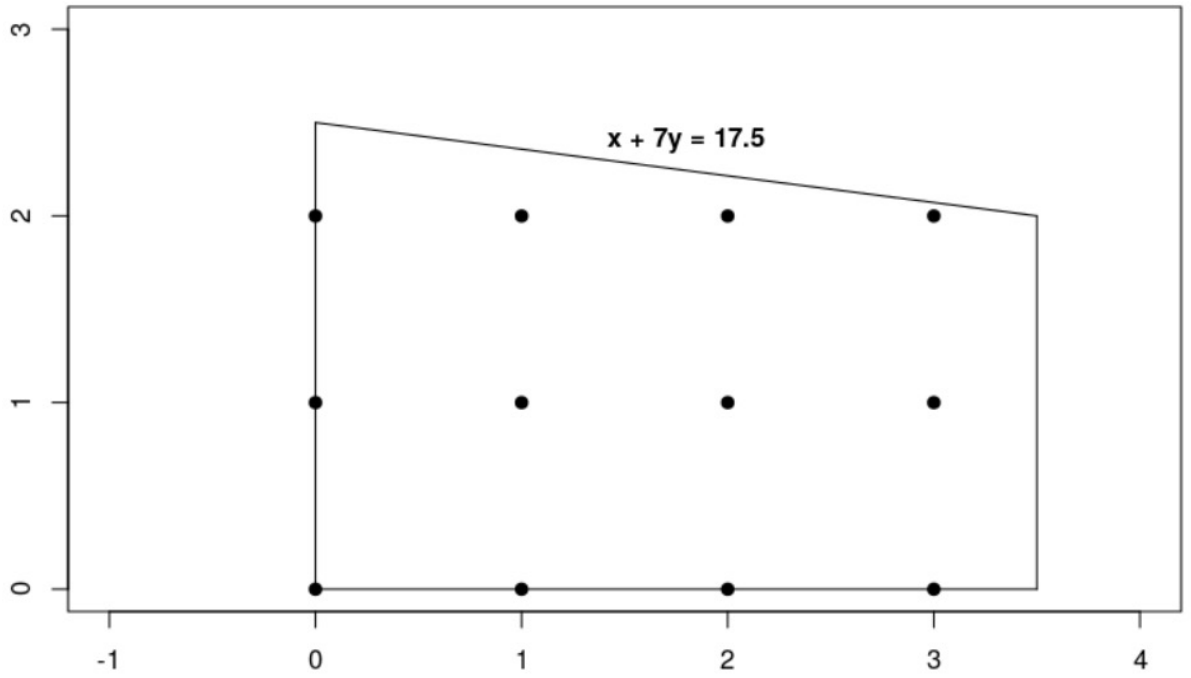


Mixed-Integer optimization problem in Java

A **Mixed-Integer optimization** example:

- **Maximize** $x + 10y$ subject to the following constraints:

$x + 7y$	≤	17.5
x	≤	3.5
x	≥	0
y	≥	0
x, y integers		



Since the constraints are linear, this is ***just a linear optimization problem in which the solutions are required to be integers!***

Mixed-Integer optimization problem in Java

Main 5 steps to solve the problem with OR-Tools:

1. Import the linear solver wrapper;
2. Declare the **MIP Solver;**
3. Define variables;
4. Define the constraints;
5. Define the objective;
6. Invoke the MIP Solver and display results.

Mixed-Integer optimization problem in Java

1. **Import the required libraries:**

```java
import com.google.ortools.Loader;
import com.google.ortools.linearsolver.MPConstraint;
import com.google.ortools.linearsolver.MPObjective;
import com.google.ortools.linearsolver.MPSolver;
import com.google.ortools.linearsolver.MPVariable;
```

2. **Declare the MIP Solver:**

```java
// Create the linear solver with the SCIP backend.
MPSolver solver = MPSolver.createSolver("SCIP");
if (solver == null) {
  System.out.println("Could not create solver SCIP");
  return;
}
```

3. **Create Variables**

```java
double infinity = java.lang.Double.POSITIVE_INFINITY;
// x and y are integer non-negative variables.
MPVariable x = solver.makeIntVar(0.0, infinity, "x");
MPVariable y = solver.makeIntVar(0.0, infinity, "y");

System.out.println("Number of variables = " + solver.numVariables());
```

MakeIntVar method allow to create variables x and y that take non-negative integer values

Mixed-Integer optimization problem in Java

4. Define the Constraints

```
// x + 7 * y <= 17.5.
MPConstraint c0 = solver.makeConstraint(-infinity, 17.5, "c0");
c0.setCoefficient(x, 1);
c0.setCoefficient(y, 7);

// x <= 3.5.
MPConstraint c1 = solver.makeConstraint(-infinity, 3.5, "c1");
c1.setCoefficient(x, 1);
c1.setCoefficient(y, 0);

System.out.println("Number of constraints = " + solver.numConstraints());
```

Constraints:

- ***C0***: x+7y <= 17.5;
- ***C1***: x <= 3.5.

5. Define the Objective of the function

```
// Maximize x + 10 * y.
MPObjective objective = solver.objective();
objective.setCoefficient(x, 1);
objective.setCoefficient(y, 10);
objective.setMaximization();
```

The following code defines the objective function, $x + 10y$, and specifies that this is a **maximization** problem

Mixed-Integer optimization problem in Java

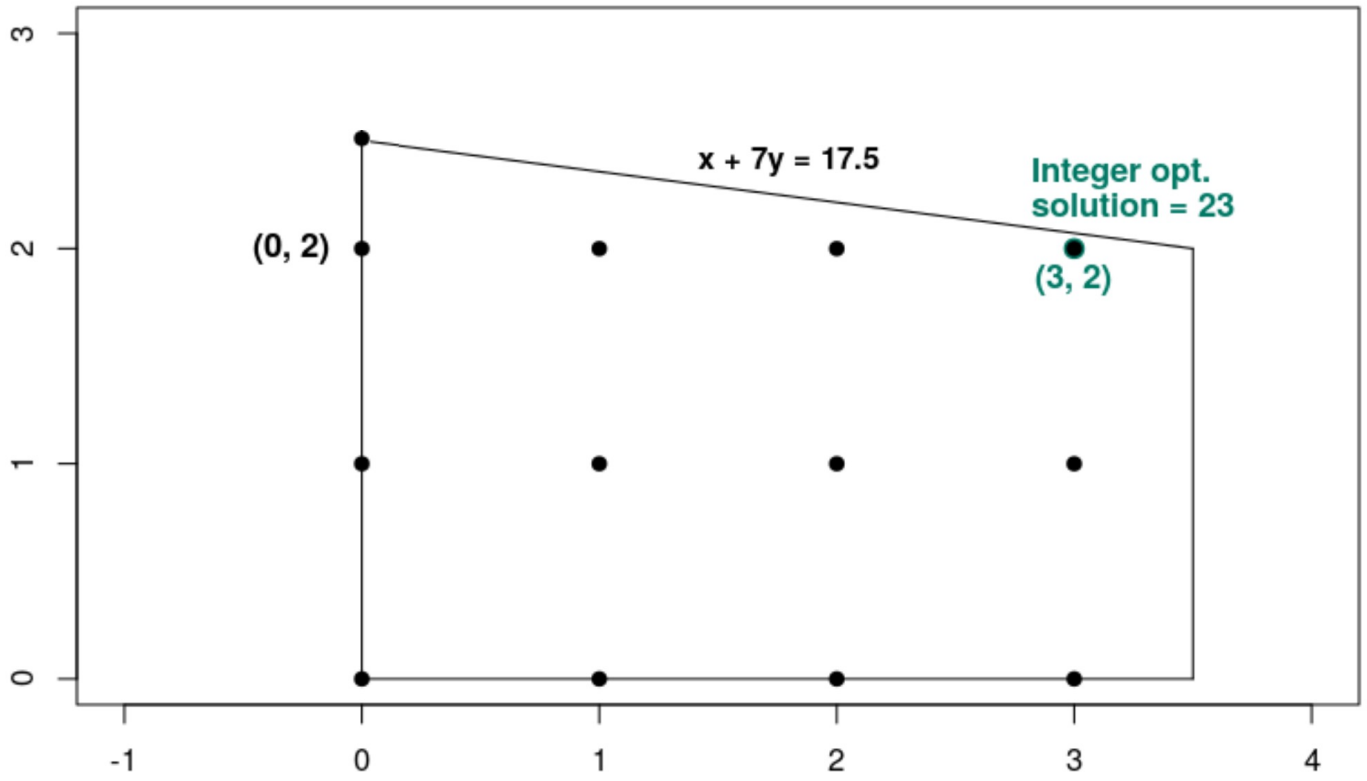
4. Invoke the solver

```
final MPSolver.ResultStatus resultStatus = solver.solve();
```

5. Display results

```
if (resultStatus == MPSolver.ResultStatus.OPTIMAL) {
  System.out.println("Solution:");
  System.out.println("Objective value = " + objective.value());
  System.out.println("x = " + x.solutionValue());
  System.out.println("y = " + y.solutionValue());
} else {
  System.err.println("The problem does not have an optimal solution!")
}
```

Mixed-Integer optimization problem in Java



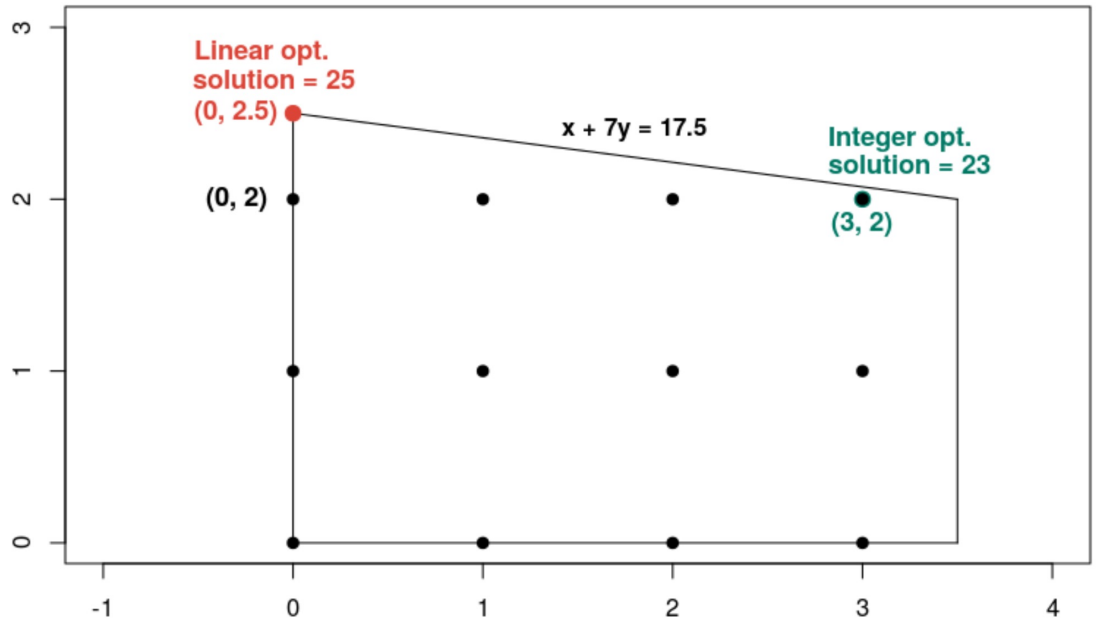
The optimal value of the objective function is **23**, which occurs at the point x = 3, y = 2.

```
Number of variables = 2
Number of constraints = 2
Solution:
Objective value = 23
x = 3
y = 2
```

Comparing Linear and Integer Optimization

Let's **compare** the solution to the integer optimization problem, with the solution to the corresponding linear optimization problem, **in which integer constraints are removed.**

1. Replace the **MIP Solver** with **LP Solver;** (SCIP -> GLOP)
2. Replace **Integer Variables** with **Continuous Variables;** (makeIntVar -> makeNumVar)
3. Check differences!



Results:

MIP:

```
Objective value = 23
x = 3
y = 2
```

LP:

```
Objective value = 25.000000
x = 0.000000
y = 2.500000
```

The integer solution is not close to the linear solution. The solutions to a linear optimization problem and the corresponding integer optimization **problems can be far apart**. Because of this, the two types of problems require different methods for their solution.

Next times..

OR-Tools: Constraints Programming examples:

- Hello world example
- SAT solver for Boolean Satisfiability Problem

OR-Tools: Routing examples:

- Simple Vehicle Routing Problem example
- Vehicle Routing Problem with capacity constraints
- ...