Computing Performance when Solving Routing Problems: From Python to Julia

Xabier Andoni Martín Solano
xamarsol@upv.es
Dept. of Applied Statistics and Operational Research, and Quality
Universitat Politècnica de València, Campus de Alcoy, Spain

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

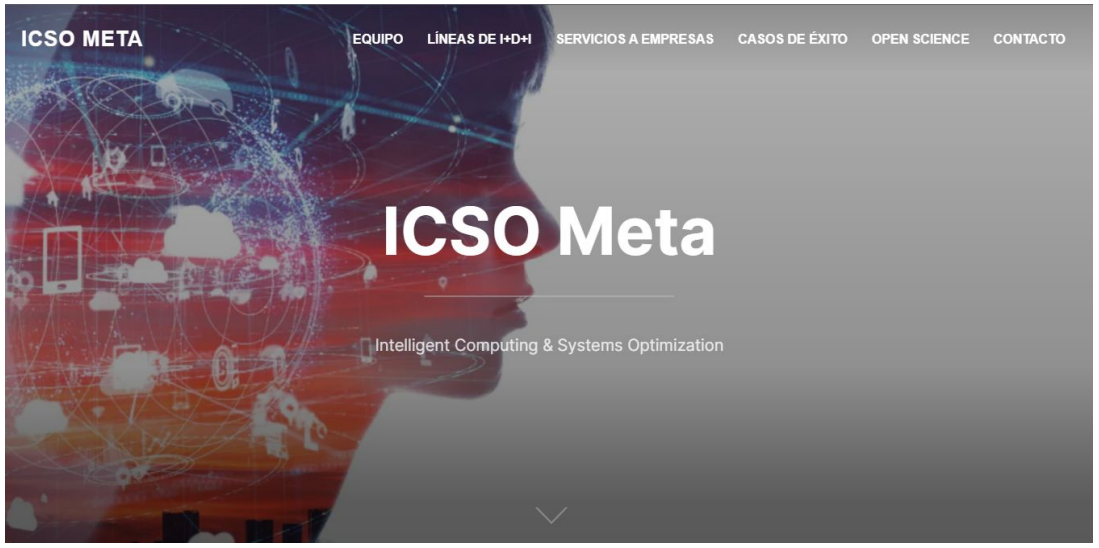
LLEIDA TECH
ARTIFICIAL INTELLIGENCE & OPTIMIZATION CONGRESS

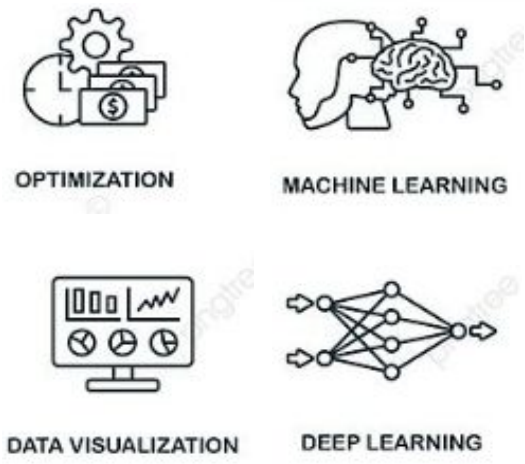
CIGIP

Part I:

Introduction

ICSO Meta





UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Universitat Oberta de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

UAB Universitat Autònoma de Barcelona

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

esade UNIVERSIDAD RAMON LLULL

UNIVERSITAT JAUME I

Universitat de Lleida

Euncet Business School

Programming Language History

- Powerful computational capabilities
- Challenging to use for non-computer science experts

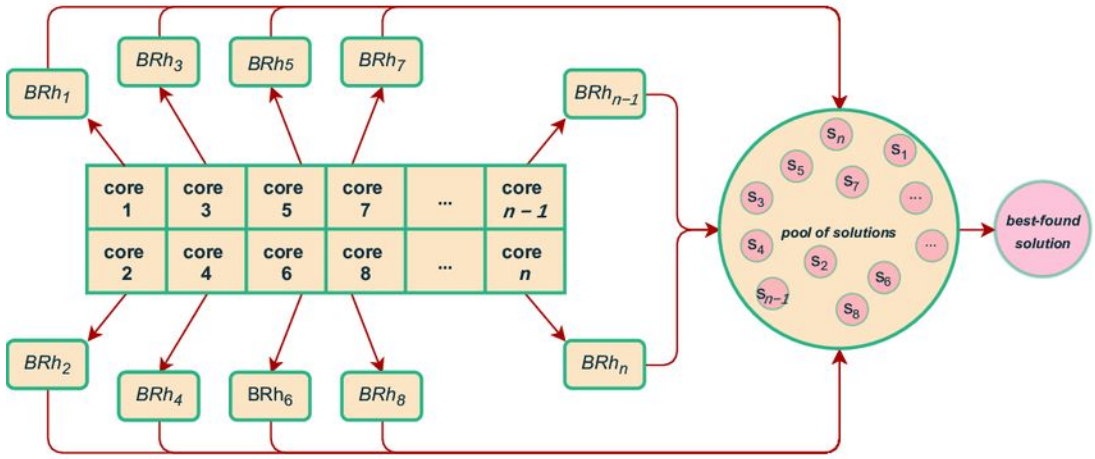
- Gentle learning curve
- Great accessibility for non-computer science experts
- Extensive environment and libraries

- High-performance language for scientific computing
- Exceptional speed and innate support for parallel computing

Motivation

- Adapt to real-world scenarios
- Conditions of an optimization problem are dynamic and can change over time

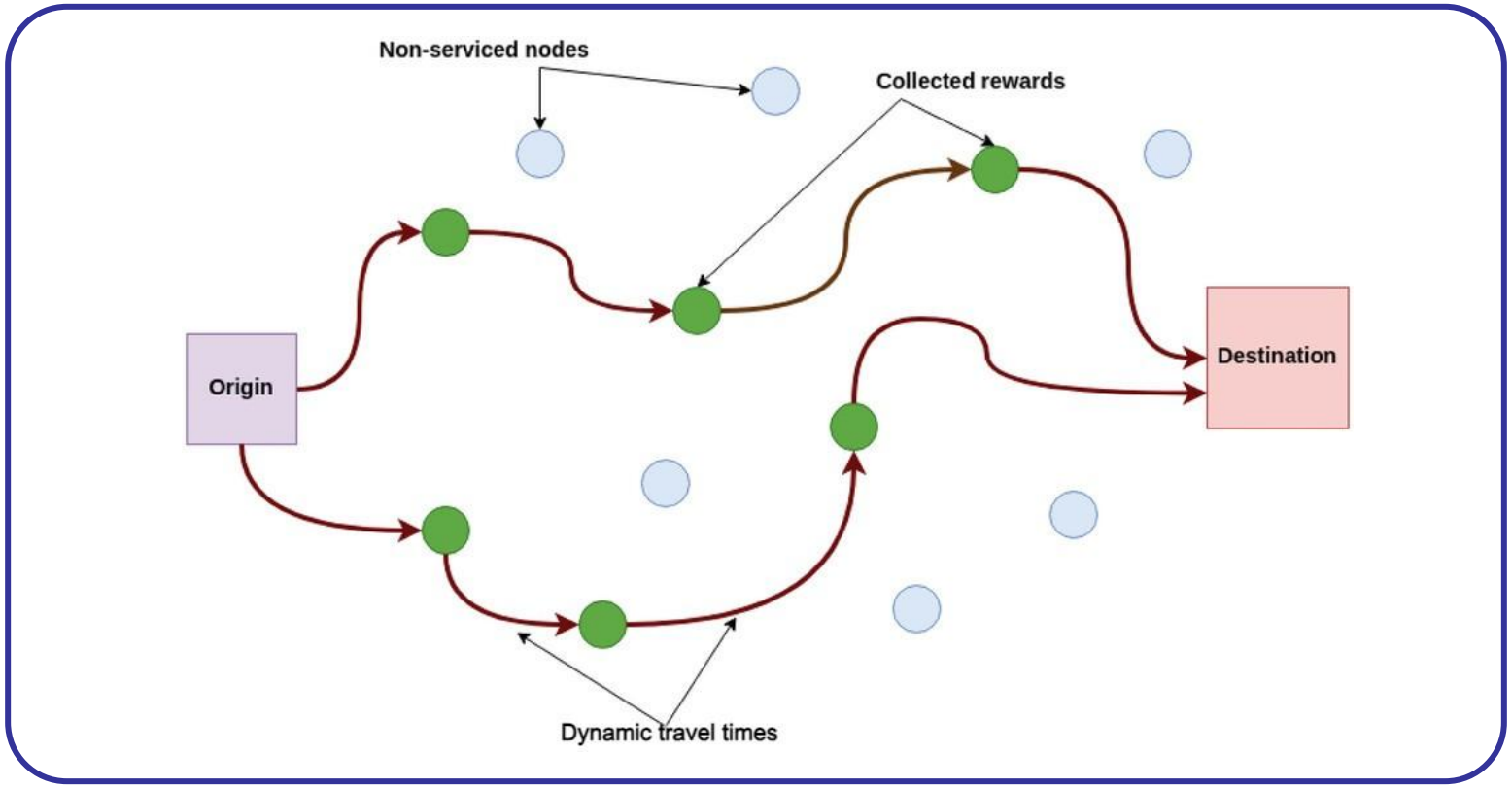
- Solved by 'agile' optimization algorithms
- The relevance of dynamism gives importance to performance, efficiency, and responsiveness



Part II:

Solving Routing Problems

The Team Orienteering Problem



- Essential problem in logistics and transportation, which finds applications across many industries
- Among the real-world applications we can find public transport, waste collection, carsharing, and ridesharing among others

The Python Approach (I)

```
class Node:

        def __init__(self, ID, x, y, reward):
                self.ID = ID
                self.x = x
                self.y = y
                self.reward = reward
                self.dnEdge = None
                self.ndEdge = None
                self.inRoute = None
                self.isLinkedToStart = False
                self.isLindedToFinish = False

class Edge:

        def __init__(self, origin, end):
                self.origin = origin
                self.end = end
                self.cost = 0.0
                self.savings = 0.0
                self.efficiency = 0.0
                self.invEdge = None

class Route:

        def __init__(self):
                self.edges = []
                self.cost = 0.0
                self.reward = 0.0
...
```

- Simple POD types in OOP style
- Dynamic typing with default values

The Python Approach (II)

```
def PJS_Heuristic(inputs, test, efficiencyList, isRand, rng):
        # Construct a dummy solution
        solution = genDummySolution(inputs)
        effList = efficiencyList.copy()

        while len(effList) > 0:
            # Select ijEdge from the savings list
            pos = getRandPos(test.betaMin, test.betaMax, len(effList), rng) if isRand else 0
            ijEdge = effList.pop(pos)

            # Determine nodes and routes of ijEdge
            iNode = ijEdge.origin
            jNode = ijEdge.end
            iRoute = iNode.inRoute
            jRoute = jNode.inRoute

            # If conditions are satisfied, merge routes and delete edge
            if isMergePossible(iNode, jNode, iRoute, jRoute, ijEdge, inputs.Tmax):
                jiEdge = ijEdge.invEdge
                if jiEdge in effList:
                    effList.remove(jiEdge)

                # Remove iEdge from iRoute
                iEdge = iRoute.edges[-1]
                iRoute.edges.remove(iEdge)
                iRoute.cost -= iEdge.cost
                iNode.isLinkedToFinish = False

                # Remove jEdge from jRoute
                jEdge = jRoute.edges[0]
                jRoute.edges.remove(jEdge)
                jRoute.cost -= jEdge.cost
                jNode.isLinkedToStart = False
```

- **Automatic memory management**
- **Use of built-in types methods for logic operations (OOP)**

The Julia Approach (I)

```
abstract type AbstractEdge end
abstract type AbstractRoute end

mutable struct Node
        id :: Int64
        x :: Float64
        y :: Float64
        reward :: Float64
        dnEdge :: Union{Nothing, AbstractEdge} = nothing
        ndEdge :: Union{Nothing, AbstractEdge} = nothing
        inRoute :: Union{Nothing, AbstractRoute} = nothing
        isInterior :: Bool = false
        isLinkedToStart :: Bool = false
        isLinkedToFinish :: Bool = false
end

mutable struct Edge <: AbstractEdge
        origin :: Node
        finish :: Node
        cost :: Float64 = 0.0
        savings :: Float64 = 0.0
        efficiency :: Float64 = 0.0
        invEdge :: Union{Nothing, AbstractEdge} = nothing
end

mutable struct Route <: AbstractRoute
        edges :: Vector{Edge} = []
        cost :: Float64 = 0.0
        reward :: Float64 = 0.0
end
...
```

- **Simple POD types in procedural style.**
- **Static typing with default values**

The Julia Approach (II)

```
function PJS_Heuristic(inputs, test, nodes, efficiencyList, isRand, rng)
        # Construct a dummy solution
        solution = genDummySolution(nodes, inputs.Tmax)
        effList = copy(efficiencyList)

        while length(effList) > 0
                # Select ijEdge from the savings list
                pos = isRand ? getRandPos(test.betaMin, test.betaMax, length(effList), rng) : 1
                ijEdge = popat!(effList, pos)

                # Determine nodes and routes of ijEdge
                iNode = ijEdge.origin
                jNode = ijEdge.finish
                iRoute = iNode.inRoute
                jRoute = jNode.inRoute

                # If conditions are satisfied, merge routes and delete edge
                if isMergeFeasible(iNode, jNode, iRoute, jRoute, ijEdge, inputs.Tmax)
                        jiEdge = ijEdge.invEdge
                        if jiEdge in effList
                                remove!(effList, jiEdge)
                        end

                        # Remove iEdge from iRoute
                        iEdge = iRoute.edges[-1]
                        remove!(iRoute.edges, iEdge)
                        iRoute.cost -= iEdge.cost
                        iNode.isLinkedToFinish = false

                        # Remove jEdge from jRoute
                        jEdge = jRoute.edges[1]
                        remove!(jRoute.edges, jEdge)
                        jRoute.cost -= jEdge.cost
                        jNode.isLinkedToStart = false
```

- **Automatic memory management**
- **Use of free functions for logic operations (procedural)**

Part III:

Computing Performance

Benchmark Process (I)

- Our benchmark process was comprehensive, testing the implementations with instances from Chao et. al (1996)
- For each instance, we conducted 1000 runs of each algorithm, meticulously collecting essential statistics
- This included calculating the mean execution time, and tracking the maximum and minimum execution times

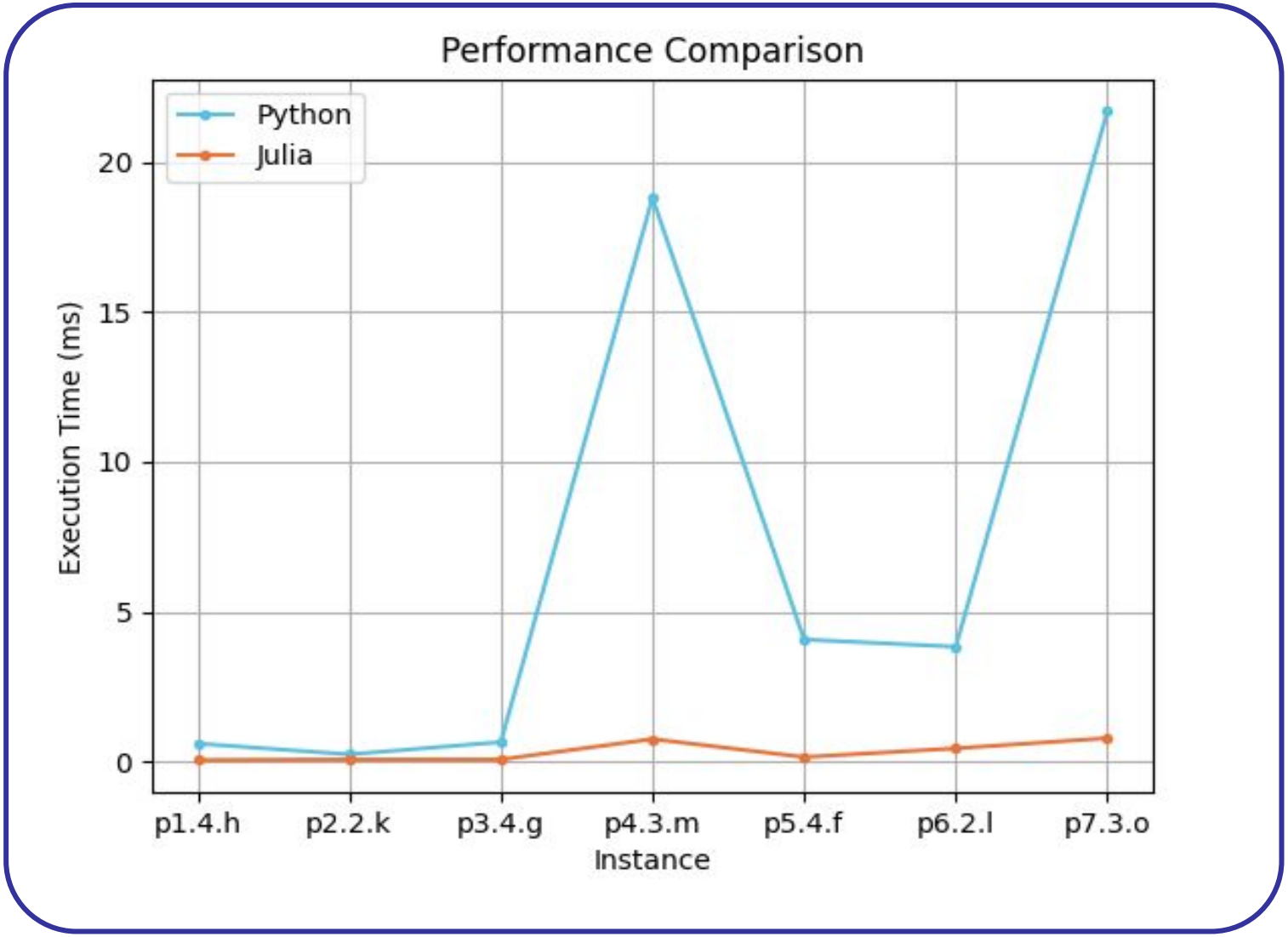
- The runtime for each instance was measured from the start of the algorithm to the moment the final solution was produced
- The PJS heuristic algorithm has been implemented using Python 3.10, and Julia 1.8.2
- Tested on a workstation with a multi-core processor Intel Xeon E5-2630 v4 and 32 GB of RAM

Benchmark Process (II)

	Python			Julia			
Instance	Avg. Time	Min. Time	Max. Time	Avg. Time	Min. Time	Max. Time	Speedup
p1.4.h	0.60	0.58	0.82	0.05	0.05	0.09	11.0
p2.2.k	0.24	0.23	0.35	0.07	0.06	0.07	3.5
p3.4.g	0.65	0.63	0.86	0.07	0.07	0.14	8.7
p4.3.m	18.80	18.21	21.52	0.75	0.71	0.84	25.1
p5.4.f	4.07	3.96	5.02	0.15	0.14	0.17	26.9
p6.2.l	3.83	3.72	4.73	0.44	0.42	0.44	8.7
p7.3.o	21.69	21.09	27.31	0.78	0.74	0.83	27.8
Average:	7.13	6.92	8.66	0.33	0.31	0.37	16.0

- Execution time of Julia implementation is an order of magnitude faster
- Most challenging instances offer the most speedup gains
- Across all instances tested, the Julia implementation delivers an average speedup factor of 16x

Benchmark Process (II)

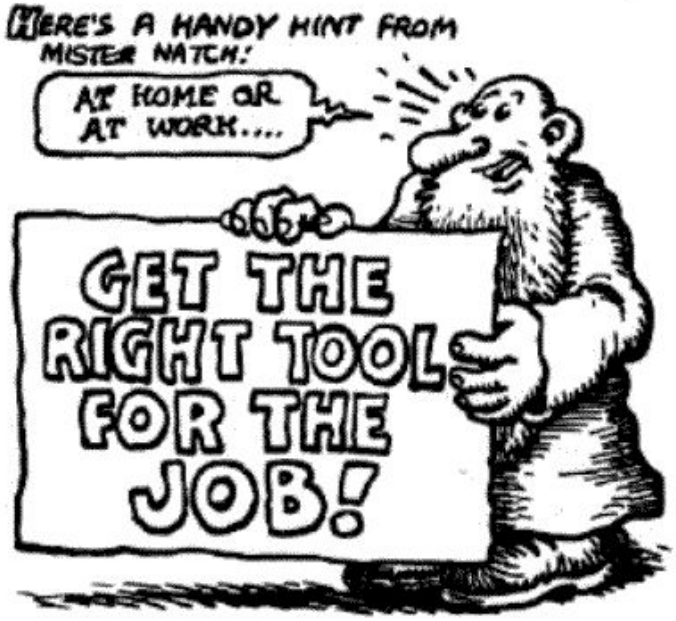


Part IV:

Conclusions and Future Directions

Conclusions

- **Powerful programming language for solving routing problems**
- **Offers a gentle learning curve for Python users**
- **Notable performance gains achieved through compiler optimizations**



Future Directions

Julia 1.9 Highlights

9 May 2023 | **The Julia contributors**

After 3 betas and 3 release candidates, Julia version 1.9 has finally(!) been released. We would like to thank all the contributors to this release and all the testers that helped with finding regressions and issues in the pre-releases. Without you, this release would not have been possible.

The full list of changes can be found in the NEWS file, but here we'll give a more in-depth overview of some of the release highlights.

1. Caching of native code
2. Package extensions
3. Heap snapshot
4. Memory usage hint for the GC with `--heap-size-hint`
5. Sorting performance
6. Tasks and the interactive thread pool
7. REPL
    1. Contextual module REPL
    2. Numbered prompt
8. DelimitedFiles – first stdlib to be upgradable

Python 3.12: A Game-Changer in Performance and Efficiency

Exploring the Exciting New Additions, Updates, and Changes in Python 3.12

AI TutorMaster · Follow

Published in Python in Plain English · 7 min read · Feb 14

497 5

> *Python 3.12, the latest version of the Python programming language, is expected to bring major optimizations and improvements to the language, with a focus on enhancing the speed, performance, and stability of the interpreter. This new release is designed to make Python a more powerful and efficient tool for developers, particularly for large and complex applications.*

Improved Multi-Threaded Parallelism

Mojo 🔥 — the programming language for all AI developers.

Mojo combines the usability of Python with the performance of C, unlocking unparalleled programmability of AI hardware and extensibility of AI models.



Thank you for your attention!

Xabier Andoni Martín Solano
xamarsol@upv.es
Dept. of Applied Statistics and Operational Research, and Quality
Universitat Politècnica de València, Campus de Alcoy, Spain

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

LLEIDA TECH
ARTIFICIAL INTELLIGENCE & OPTIMIZATION CONGRESS

CIGIP