Automatic Conference Scheduling with PuLP

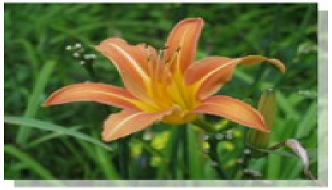
EuroPython 2017
Rimini, Italy

Marc-André Lemburg :: eGenix.com GmbH

Speaker Introduction

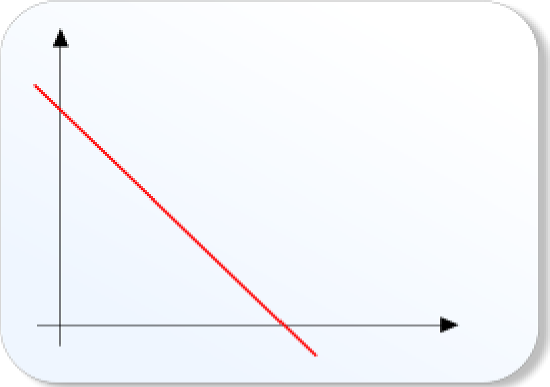
Marc-André Lemburg

- Python since 1994
- Studied Mathematics
- eGenix.com GmbH
- Senior Software Architect
- Consultant / Trainer
- Python Core Developer
- EuroPython Society
- Python Software Foundation
- Based in Düsseldorf, Germany



Linear Programming

- Term from "Operations Research" in mathematics
  - "Programming" means:
    find an optimal solution for a planing problem
  - "Linear", because only linear relationships are addressed, e.g. y = a*x + b
- Careful:
  - Problem usually easy to understand
  - Finding solutions can be very hard



Linear Programming

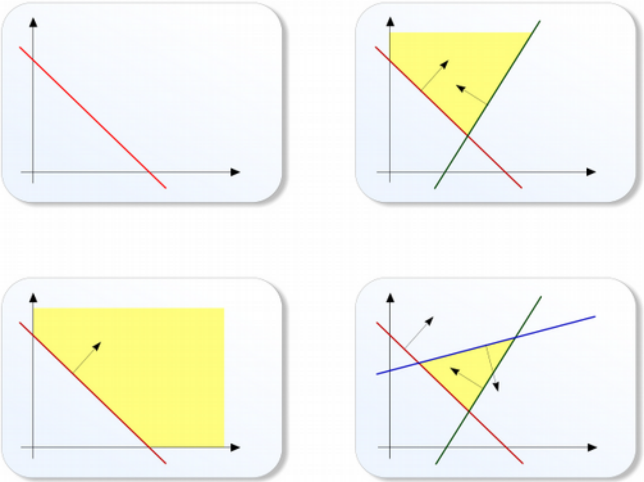
- Integer Programming
  - Additional restriction:
    values may only be integers, not floats
  - In practice: often a mix of linear + integer programming
  - Often: exponential runtime
- Examples
  - Knapsack problem
  - Traveling salesman problem
  - Project optimization (dependencies, resources)
  - Conference Scheduling

Linear Programming

- Mathematics
  - Variables: x_1, x_2, …, x_n (float or integer)
  - Linear target function (objective)
  - Rules for valid solutions (constraints) of the form:

$$a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots a_n x_n \{<=, =, >=\} b$$

with constants a_i and b



  - Goal: find an (optimal) solution x, which fulfills all constraints and minimizes (or maximizes) the objective

PuLP: A COIN-OR project

- COIN-OR
  - Library for operations research
  - PuLP is a Python Interface for the LP part of COIN-OR
  - COIN-OR comes with a few LP solvers
  - http://www.coin-or.org/
- PuLP – LP Solver Front-End
  - Standardized interface for LP solvers
  - Comes with a slow solver (great for development)
  - Faster: GLPK (GNU LP Kernel)
  - Other commercial solvers: CPLEX, GUROBI
  - https://projects.coin-or.org/PuLP

PuLP: Datatypes

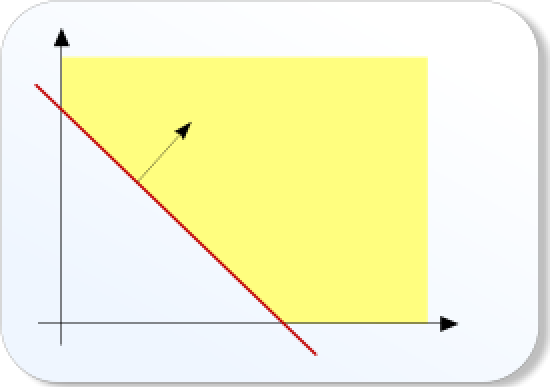
- LpProblem
  - Defines the LP problem
  - Holds the constraints and objective function
  - Interface to the LP Solver (external)
- LpVariable
  - Abstracts an LP variable (with name)
  - Values will be changed by the solver
  - Float or integer
  - Defines the permitted variable value range

PuLP: Datatypes

- LpConstraint – Constraint rule
  - Form: $a_1x_1 + a_2x_2 + a_3x_3 + ...a_nx_n\{<=, =, >=\}b$

    affine function OP numeric term
    OP can be one of: <=, =, >=

  - Can have a name (for debugging)



… some more (e.g. elastic constraints)

PuLP: Documentation

- Not that great :-(
  - Package documentation:
    https://pythonhosted.org/PuLP/index.html
    Incomplete, misses details.
  - Source code:
    https://github.com/coin-or/pulp/blob/master/src/pulp/
  - Some blog posts:
    https://scaron.info/blog/linear-programming-in-python-with-pulp.html
    http://benalexkeen.com/linear-programming-with-python-and-pulp/

PuLP: Example Conference Scheduling

- Inspiration:
  - Talk from David MacIver at PyCon UK 2016
    https://www.youtube.com/watch?v=OkusHEBOhmQ

- Use case:
  - Help with scheduling EuroPython 2017 or later

PuLP: Example Conference Scheduling

- Goal:
  - Simplify scheduling
  - Optimize speaker (and attendee) satisfaction
- Constraints:
  - Multiple rooms of different sizes
  - Talk slots of varying lengths (e.g. 30min / 45min)
  - Talks with varying lengths
  - Speakers cannot give two talks at the same time
  - Speakers may have availability constraints

Conference data: rooms and talks

```
import pulp
import sys

# Rooms and sizes
rooms = {
    'A': 100,
    'B': 50,
    'C': 50,
}

# Talks and duration
talks = {
    'Introduction to Python': 30,
    'Introduction to JavaScript': 30,
    'Introduction to C#': 60,
    'Introduction to Java': 90,
    'Introduction to C': 90,
}
```

Conference data: talk slots

```
# Available slots: room, start times and durations; total time: 150 min
slots = [
        ('A', 0, 30),
        ('A', 30, 60),
        ('A', 90, 60),
        ('B', 0, 30),
        ('B', 30, 90),
        ('B', 120, 30),
        ('C', 0, 90),
        ('C', 90, 60),
]
```

LP problem and variables

```
### Problem definition

problem = pulp.LpProblem('Conference Schedule', sense=pulp.LpMaximize)

### Variables

# Slot assignment variables
assign = {
    (talk, slot): pulp.LpVariable('%r in slot %r' %
                                  (talk, slot),
                                  cat=pulp.LpBinary)
    for talk in talks
    for slot in slots
}
#print (assign)
```

Constraints for talk slots

```
### Constraints

# Add slot constraints
for slot in slots:

    # Each slot may only be assigned at most once
    problem.addConstraint(
        sum(assign[(talk, slot)]
            for talk in talks)
        <= 1)
```

Only assign one talk per slot

Constraints for talks

We need to assign all talks

```
# Add talk constraints
for talk in talks:

    # All talks have to be assigned
    problem.addConstraint(
        sum(assign[(talk, slot)]
            for slot in slots)
        == 1)

    # Talk durations must fit slots
    problem.addConstraint(
        sum(slot[2] * assign[(talk, slot)]
            for slot in slots)
        == talks[talk])
```

Talks must fit the talk slots

Special constraints

Speaker not always available

```
### Special requirements

# Talk must start later
talk = 'Introduction to Python'
for slot in slots:
    if slot[1] < 90:
        problem.addConstraint(
            assign[(talk, slot)] == 0)
```

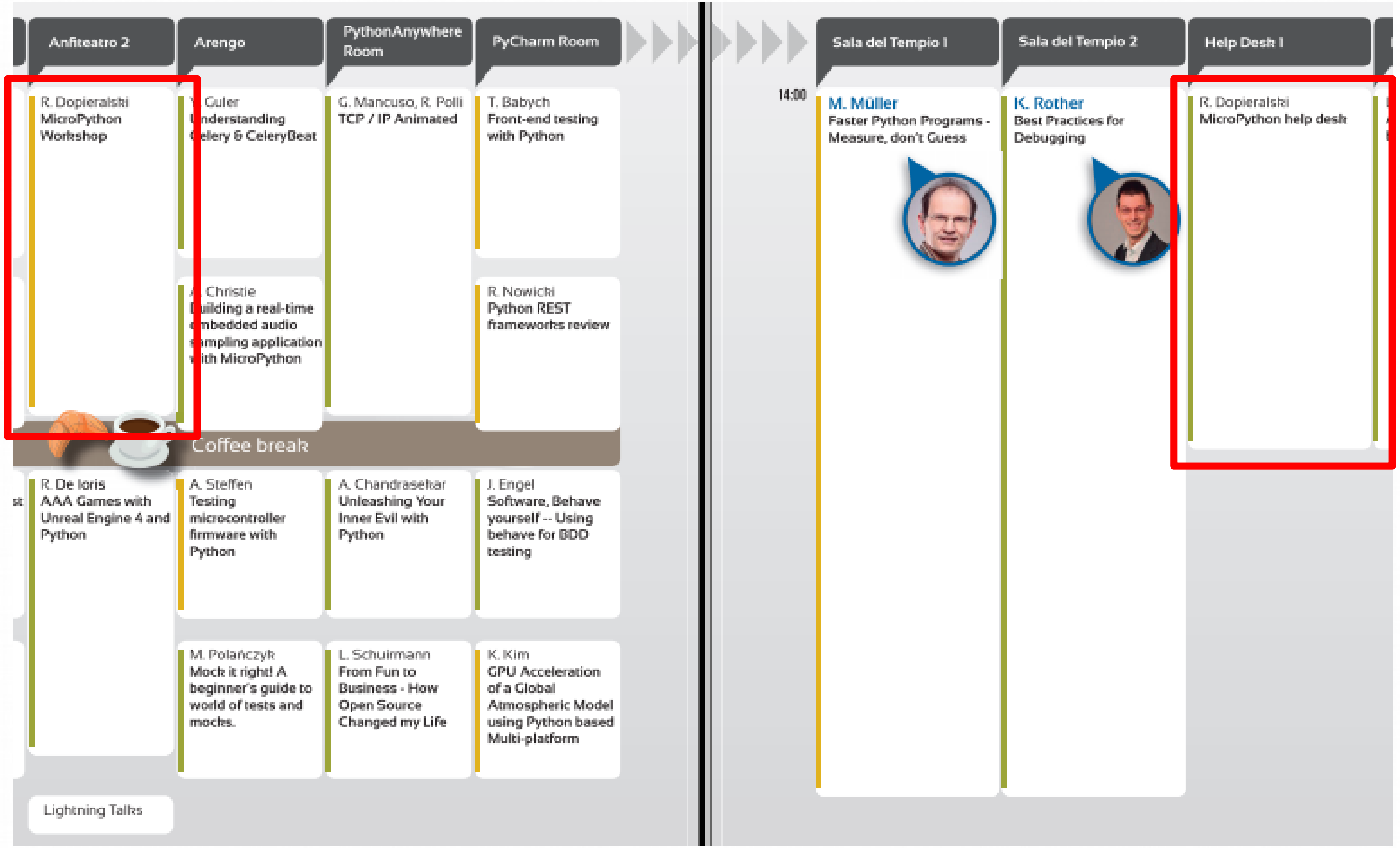
More problems: How to prevent overlaps

```
# Two talks given by the same person
talks_same_speaker = (
    'Introduction to C#',
    'Introduction to Java',
    )

# Available slots: room, start times and durations; total time: 150 min
slots = [
    ('A', 0, 30),
    ('A', 30, 60),
    ('A', 90, 60),
    ('B', 0, 30),
    ('B', 30, 90),
    ('B', 120, 30),
    ('C', 0, 90),
    ('C', 90, 60),
]
```

More than one talk per speaker

Different slots per room

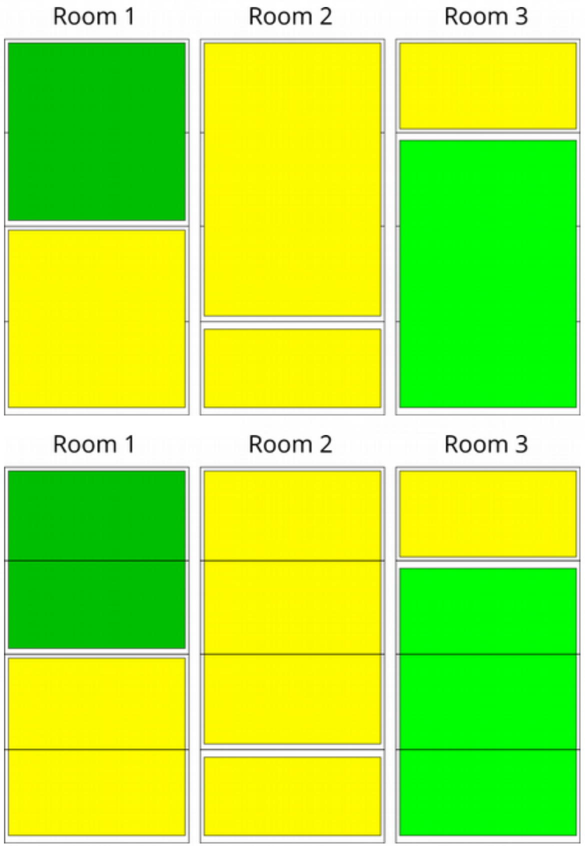
More problems: How to prevent overlaps



Solution: Divide slots into smaller standard blocks

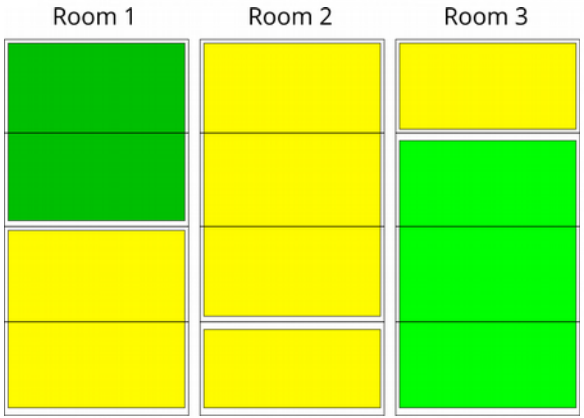
Add new variables

```
# Block assignment variables
block_assign = {
    (talk, block): pulp.LpVariable('%r in block %r' %
                                   (talk, block),
                                   cat=pulp.LpBinary)
    for talk in talks
    for block in blocks
}
```



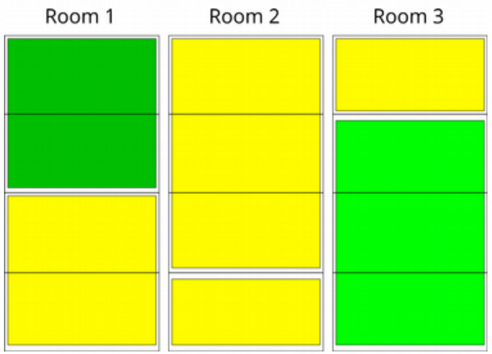
Solution: Define some helper mappings

```
# Blocks: slots are arranged in a fixed 30 min block schedule; each
# block is mapped to a slot in the schedule
blocks = {
    (room, start): None
    for room in sorted(rooms)
    for start in range(0, 150, 30)
}

# Find blocks per slot
blocks_per_slot = {}
for slot in slots:
    room, start, duration = slot
    slot_blocks = []
    for block_time in range(start, start + duration, 30):
        blocks[(room, block_time)] = slot
        slot_blocks.append((room, block_time))
    blocks_per_slot[slot] = slot_blocks

# Find blocks per start time
time_blocks = {
    start: [(room, start)
            for room in sorted(rooms)]
    for start in range(0, 150, 30)
}
```



Solution: Link slots and blocks

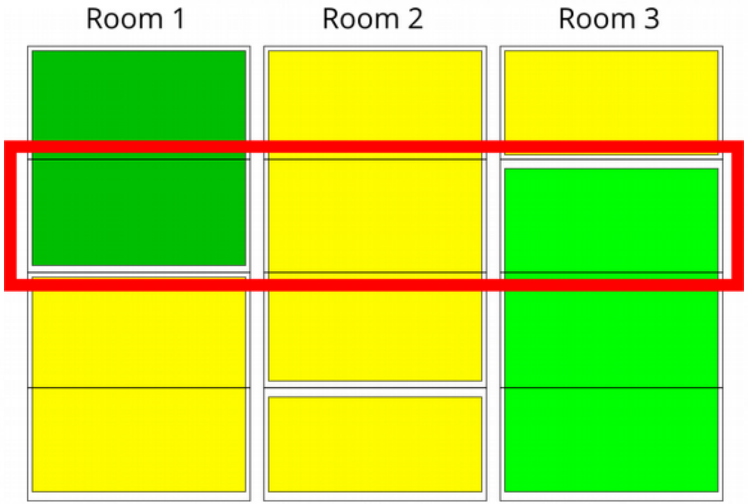
```
# Add slot constraints
for slot in slots:

    # Each slot may only be assigned at most once
    problem.addConstraint(
        sum(assign[(talk, slot)]
            for talk in talks)
        <= 1)

    # Tie the blocks to the corresponding slots
    for block in blocks_per_slot[slot]:
        for talk in talks:
            problem.addConstraint(
                assign[(talk, slot)] == block_assign[(talk, block)])
```



Constraint: More than one talk per speaker

Using blocks, you can now define the constraint:

```
# Two talks given by the same person
talks_same_speaker = (
    'Introduction to C#',
    'Introduction to Java',
    )
for blocks_per_time in time_blocks.values():
    problem.addConstraint(
        sum(block_assign[(talk, block)]
            for talk in talks_same_speaker
            for block in blocks_per_time)
        <= 1)
```



Finally: Define objective function & run solver

```
# Maximize the happiness of attendees
#problem.objective = happiness(assign)

# Solve the problem using GLPK
#solver = pulp.GLPK_CMD(msg=1)
solver = None
problem.solve(solver)

# Check status
if problem.status == 1:
    print ('Found a solution.\n')
else:
    print ('Failed to find a solution: %s' % pulp.LpStatus[problem.status])
    sys.exit(1)
```

Show the result

```
# Print the slot assignments
print ('Slot assignments:')
print ('-'*72)
for (talk, slot), assigned in sorted(assign.items()):
    if pulp.value(assigned):
        print ('Talk %r assigned to Slot %r' % (talk, slot))
print ('')

# Print the block assignments
print ('Block assignments:')
print ('-'*72)
for (talk, block), assigned in sorted(block_assign.items()):
    if pulp.value(assigned):
        print ('Talk %r assigned to Block %r' % (talk, block))
print ('')
```

Show the result: Raw data

```
Found a solution.

Slot assignments:
_________________________________________________________________________

Talk 'Introduction to C' assigned to Slot ('B', 30, 90)
Talk 'Introduction to C#' assigned to Slot ('A', 90, 60)
Talk 'Introduction to Java' assigned to Slot ('C', 0, 90)
Talk 'Introduction to JavaScript' assigned to Slot ('B', 0, 30)
Talk 'Introduction to Python' assigned to Slot ('B', 120, 30)

Block assignments:
_________________________________________________________________________

Talk 'Introduction to C' assigned to Block ('B', 30)
Talk 'Introduction to C' assigned to Block ('B', 60)
Talk 'Introduction to C' assigned to Block ('B', 90)
Talk 'Introduction to C#' assigned to Block ('A', 90)
Talk 'Introduction to C#' assigned to Block ('A', 120)
Talk 'Introduction to Java' assigned to Block ('C', 0)
Talk 'Introduction to Java' assigned to Block ('C', 30)
Talk 'Introduction to Java' assigned to Block ('C', 60)
Talk 'Introduction to JavaScript' assigned to Block ('B', 0)
Talk 'Introduction to Python' assigned to Block ('B', 120)
```

Show the result: As Schedule

```
Room schedule:
________________________________________________________________________

Room A, Time   90: Introduction to C#              (60 min)
Room B, Time    0: Introduction to JavaScript      (30 min)
Room B, Time   30: Introduction to C               (90 min)
Room B, Time  120: Introduction to Python          (30 min)
Room C, Time    0: Introduction to Java            (90 min)

Talk listing:
________________________________________________________________________

Introduction to C               (90 min):
    Room B, Time   30 (90 min)
Introduction to C#              (60 min):
    Room A, Time   90 (60 min)
Introduction to Java            (90 min):
    Room C, Time    0 (90 min)
Introduction to JavaScript      (30 min):
    Room B, Time    0 (30 min)
Introduction to Python          (30 min):
    Room B, Time  120 (30 min)
```

More possibilities

- Use room capacities and attendee preferences
- Add tracks:
  Group talks by topic (preferably in a single room)
- When having to apply changes after publication of the schedule (new constraints, speaker cancellations):

  Minimize changes

Difficulty: Finding a suitable model

- LP only supports linear combinations
  - Constraints of the form $x_i * x_j$ are not supported
  - Dynamic dependencies are hard to model
  - "Programming" is declarative (as in e.g. SQL), not imperative (as in e.g. Python)

- Models have great influence on runtime

PuLP: Summary of gotchas

- LpVariable *var* doesn't always behave like a Python number
  - LpBinary variables can assume values outside their valid value range {0, 1} during solving
    better: test for *(var > 0)*
  - *if var:* is always true, when not running in the solver
    better: *if pulp.value(var):*
- LpProblem can fail to deliver a result
  - *assert problem.status == 1*

- Conclusion: Always test your solver !

Faster than PuLP

- CVXOPT - Python software for convex optimization
  - http://cvxopt.org/
  - Uses a different API than PuLP
  - Much better documentation
  - Up to 10-70x faster than PuLP
    https://scaron.info/blog/linear-programming-in-python-with-cvxopt.html
- CVXPY - Python-embedded modeling language for convex optimization problems
  - http://www.cvxpy.org/
- PICOS - Python Interface for conic optimization solvers
  - http://picos.zib.de/

Conference scheduling using Python

- PyCon UK: Conference Scheduler
  - https://github.com/PyconUK/ConferenceScheduler
  - Documentation: http://conference-scheduler.readthedocs.io/
  - Fairly new: only 2 months old
  - Uses PuLP, completely automated
- Alexander tried to use it for EuroPython 2017: failed due to exponential runtime

Conference scheduling using Python

- EuroPython 2017 Scheduler
  - Written by Alexander Hendorf
  - Doesn't use PuLP, but a similar model to the PyCon UK one

  - Works based on clustering + random shuffling + human touch

Thank you for your attention !

Beautiful is better than ugly.

Contact

eGenix.com Software, Skills and Services GmbH

Marc-André Lemburg

Pastor-Löh-Str. 48

D-40764 Langenfeld

Germany

eMail:	mal@egenix.com
Phone:	+49 211 9304112
Fax:	+49 211 3005250
Web:	http://www.egenix.com/

Automatic Conference Scheduling with PuLP

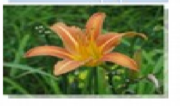
EuroPython 2017
Rimini, Italy

Marc-André Lemburg :: eGenix.com GmbH

Speaker Introduction

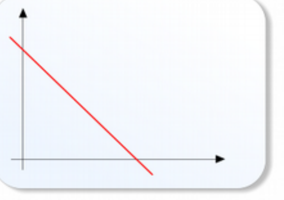
Marc-André Lemburg

- Python since 1994
- Studied Mathematics
- eGenix.com GmbH
- Senior Software Architect
- Consultant / Trainer
- Python Core Developer
- EuroPython Society
- Python Software Foundation
- Based in Düsseldorf, Germany



Linear Programming

- Term from “Operations Research” in mathematics
  - “Programming” means:
    find an optimal solution for a planing problem
  - “Linear”, because only linear relationships
    are addressed, e.g. y = a*x + b
- Careful:
  - Problem usually easy to understand
  - Finding solutions can be very hard



Linear Programming

- Integer Programming
  - Additional restriction:
    values may only be integers, not floats
  - In practice: often a mix of linear + integer programming
  - Often: exponential runtime
- Examples
  - Knapsack problem
  - Traveling salesman problem
  - Project optimization (dependencies, resources)
  - Conference Scheduling

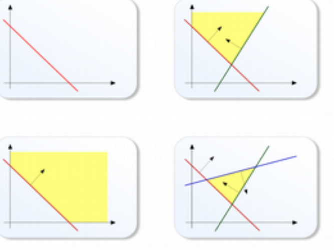
Linear Programming

- Mathematics
  - Variables: x_1, x_2, …, x_n (float or integer)
  - Linear target function (objective)
  - Rules for valid solutions (constraints) of the form:

    $$a_1x_1 + a_2x_2 + a_3x_3 + ...a_nx_n\{<=, =, >=\}b$$

    with constants a_i and b

  - Goal: find an (optimal) solution x, which fulfills all constraints and minimizes (or maximizes) the objective



PuLP: A COIN-OR project

- COIN-OR
  - Library for operations research
  - PuLP is a Python Interface for the LP part of COIN-OR
  - COIN-OR comes with a few LP solvers
  - http://www.coin-or.org/
- PuLP – LP Solver Front-End
  - Standardized interface for LP solvers
  - Comes with a slow solver (great for development)
  - Faster: GLPK (GNU LP Kernel)
  - Other commercial solvers: CPLEX, GUROBI
  - https://projects.coin-or.org/PuLP

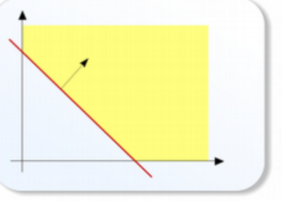
PuLP: Datatypes

- LpProblem
  - Defines the LP problem
  - Holds the constraints and objective function
  - Interface to the LP Solver (external)
- LpVariable
  - Abstracts an LP variable (with name)
  - Values will be changed by the solver
  - Float or integer
  - Defines the permitted variable value range

PuLP: Datatypes

- LpConstraint – Constraint rule
  - Form: $a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots a_n x_n \{<=,=,>=\} b$

    affine function OP numeric term
    OP can be one of: <=, =, >=
  - Can have a name (for debugging)



… some more (e.g. elastic constraints)

PuLP: Documentation

- Not that great :-(
  - Package documentation:
    https://pythonhosted.org/PuLP/index.html
    Incomplete, misses details.
  - Source code:
    https://github.com/coin-or/pulp/blob/master/src/pulp/
  - Some blog posts:
    https://scaron.info/blog/linear-programming-in-python-with-pulp.html
    http://benalexkeen.com/linear-programming-with-python-and-pulp/

PuLP: Example Conference Scheduling

- Inspiration:
  - Talk from David MacIver at PyCon UK 2016
    https://www.youtube.com/watch?v=OkusHEBOhmQ

- Use case:
  - Help with scheduling EuroPython 2017 or later

PuLP: Example Conference Scheduling

- Goal:
  - Simplify scheduling
  - Optimize speaker (and attendee) satisfaction
- Constraints:
  - Multiple rooms of different sizes
  - Talk slots of varying lengths (e.g. 30min / 45min)
  - Talks with varying lengths
  - Speakers cannot give two talks at the same time
  - Speakers may have availability constraints

Conference data: rooms and talks

```
import pulp
import sys

# Rooms and sizes
rooms = {
    'A': 100,
    'B': 50,
    'C': 50,
}

# Talks and duration
talks = {
    'Introduction to Python': 30,
    'Introduction to JavaScript': 30,
    'Introduction to C#': 60,
    'Introduction to Java': 90,
    'Introduction to C': 90,
}
```

Conference data: talk slots

```
# Available slots: room, start times and durations; total time: 150 min
slots = [
    ('A', 0, 30),
    ('A', 30, 60),
    ('A', 90, 60),
    ('B', 0, 30),
    ('B', 30, 90),
    ('B', 120, 30),
    ('C', 0, 90),
    ('C', 90, 60),
]
```

LP problem and variables

```
### Problem definition

problem = pulp.LpProblem('Conference Schedule', sense=pulp.LpMaximize)

### Variables

# Slot assignment variables
assign = {
    (talk, slot): pulp.LpVariable('%r in slot %r' %
                                  (talk, slot),
                                  cat=pulp.LpBinary)
    for talk in talks
    for slot in slots
}
#print (assign)
```

Constraints for talk slots

```
### Constraints

# Add slot constraints
for slot in slots:

    # Each slot may only be assigned at most once
    problem.addConstraint(
        sum(assign[(talk, slot)]
            for talk in talks)
        <= 1)
```

Only assign one talk per slot

Constraints for talks

We need to assign all talks

```
# Add talk constraints
for talk in talks:

    # All talks have to be assigned
    problem.addConstraint(
        sum(assign[(talk, slot)]
            for slot in slots)
        == 1)

    # Talk durations must fit slots
    problem.addConstraint(
        sum(slot[2] * assign[(talk, slot)]
            for slot in slots)
        == talks[talk])
```

Talks must fit the talk slots

Special constraints

Speaker not always available

```
### Special requirements

# Talk must start later
talk = 'Introduction to Python'
for slot in slots:
    if slot[1] < 90:
        problem.addConstraint(
            assign[(talk, slot)] == 0)
```

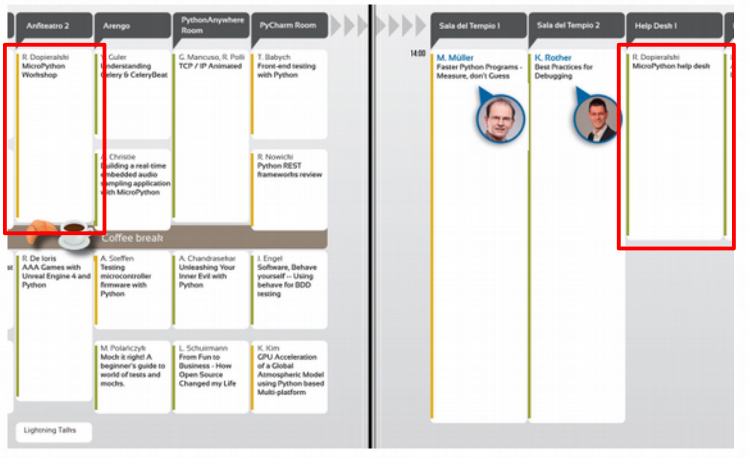
More problems: How to prevent overlaps

```
# Two talks given by the same person
talks_same_speaker = (
    'Introduction to C#',
    'Introduction to Java',
    )

# Available slots: room, start times and durations; total time: 150 min
slots = [
     ('A', 0, 30),
     ('A', 30, 60),
     ('A', 90, 60),
     ('B', 0, 30),
     ('B', 30, 90),
     ('B', 120, 30),
     ('C', 0, 90),
     ('C', 90, 60),
]
```

More than one talk per speaker

Different slots per room

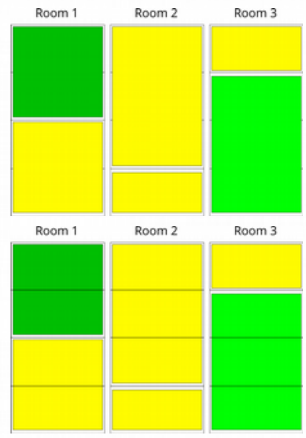
More problems: How to prevent overlaps



Solution: Divide slots into smaller standard blocks

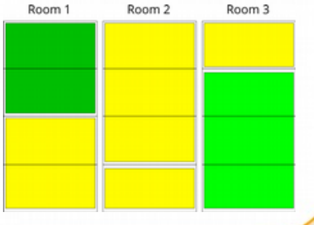
Add new variables

```
# Block assignment variables
block_assign = {
    (talk, block): pulp.LpVariable('%r in block %r' %
                                   (talk, block),
                                   cat=pulp.LpBinary)
    for talk in talks
    for block in blocks
}
```

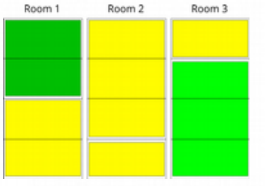


Solution: Define some helper mappings

```
# Blocks: slots are arranged in a fixed 30 min block schedule; each
# block is mapped to a slot in the schedule
blocks = {
    (room, start): None
    for room in sorted(rooms)
    for start in range(0, 150, 30)
}

# Find blocks per slot
blocks_per_slot = {}
for slot in slots:
    room, start, duration = slot
    slot_blocks = []
    for block_time in range(start, start + duration, 30):
        blocks[(room, block_time)] = slot
        slot_blocks.append((room, block_time))
    blocks_per_slot[slot] = slot_blocks

# Find blocks per start time
time_blocks = {
    start: [(room, start)
            for room in sorted(rooms)]
    for start in range(0, 150, 30)
}
```



Solution: Link slots and blocks

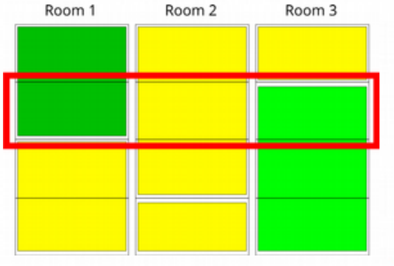



```python
# Add slot constraints
for slot in slots:

    # Each slot may only be assigned at most once
    problem.addConstraint(
        sum(assign[(talk, slot)]
            for talk in talks)
        <= 1)

    # Tie the blocks to the corresponding slots
    for block in blocks_per_slot[slot]:
        for talk in talks:
            problem.addConstraint(
                assign[(talk, slot)] == block_assign[(talk, block)])
```

Constraint: More than one talk per speaker

Using blocks, you can now define the constraint:

```
# Two talks given by the same person
talks_same_speaker = (
    'Introduction to C#',
    'Introduction to Java',
    )
for blocks_per_time in time_blocks.values():
    problem.addConstraint(
        sum(block_assign[(talk, block)]
            for talk in talks_same_speaker
            for block in blocks_per_time)
        <= 1)
```



Finally: Define objective function & run solver

```
# Maximize the happiness of attendees
#problem.objective = happiness(assign)

# Solve the problem using GLPK
#solver = pulp.GLPK_CMD(msg=1)
solver = None
problem.solve(solver)

# Check status
if problem.status == 1:
    print ('Found a solution.\n')
else:
    print ('Failed to find a solution: %s' % pulp.LpStatus[problem.status])
    sys.exit(1)
```

Show the result

```
# Print the slot assignments
print ('Slot assignments:')
print ('-'*72)
for (talk, slot), assigned in sorted(assign.items()):
    if pulp.value(assigned):
        print ('Talk %r assigned to Slot %r' % (talk, slot))
print ('')

# Print the block assignments
print ('Block assignments:')
print ('-'*72)
for (talk, block), assigned in sorted(block_assign.items()):
    if pulp.value(assigned):
        print ('Talk %r assigned to Block %r' % (talk, block))
print ('')
```

Show the result: Raw data

```
Found a solution.

Slot assignments:
------------------------------------------------------------------------
Talk 'Introduction to C' assigned to Slot ('B', 30, 90)
Talk 'Introduction to C#' assigned to Slot ('A', 90, 60)
Talk 'Introduction to Java' assigned to Slot ('C', 0, 90)
Talk 'Introduction to JavaScript' assigned to Slot ('B', 0, 30)
Talk 'Introduction to Python' assigned to Slot ('B', 120, 30)

Block assignments:
------------------------------------------------------------------------
Talk 'Introduction to C' assigned to Block ('B', 30)
Talk 'Introduction to C' assigned to Block ('B', 60)
Talk 'Introduction to C' assigned to Block ('B', 90)
Talk 'Introduction to C#' assigned to Block ('A', 90)
Talk 'Introduction to C#' assigned to Block ('A', 120)
Talk 'Introduction to Java' assigned to Block ('C', 0)
Talk 'Introduction to Java' assigned to Block ('C', 30)
Talk 'Introduction to Java' assigned to Block ('C', 60)
Talk 'Introduction to JavaScript' assigned to Block ('B', 0)
Talk 'Introduction to Python' assigned to Block ('B', 120)
```

Show the result: As Schedule

```
Room schedule:
________________________________________________________________________
Room A, Time   90: Introduction to C#            (60 min)
Room B, Time    0: Introduction to JavaScript    (30 min)
Room B, Time   30: Introduction to C             (90 min)
Room B, Time  120: Introduction to Python        (30 min)
Room C, Time    0: Introduction to Java          (90 min)

Talk listing:
________________________________________________________________________
Introduction to C              (90 min):
    Room B, Time   30 (90 min)
Introduction to C#             (60 min):
    Room A, Time   90 (60 min)
Introduction to Java           (90 min):
    Room C, Time    0 (90 min)
Introduction to JavaScript     (30 min):
    Room B, Time    0 (30 min)
Introduction to Python         (30 min):
    Room B, Time  120 (30 min)
```

More possibilities

- Use room capacities and attendee preferences
- Add tracks:
  Group talks by topic (preferably in a single room)
- When having to apply changes after publication of the schedule (new constraints, speaker cancellations):

  Minimize changes

Difficulty: Finding a suitable model

- LP only supports linear combinations
  - Constraints of the form $x_i * x_j$ are not supported
  - Dynamic dependencies are hard to model
  - “Programming” is declarative (as in e.g. SQL), not imperative (as in e.g. Python)
- Models have great influence on runtime

PuLP: Summary of gotchas

- LpVariable *var* doesn't always behave like a Python number
  - LpBinary variables can assume values outside their valid value range {0, 1} during solving
    better: test for *(var > 0)*
  - *if var:* is always true, when not running in the solver
    better: *if pulp.value(var):*
- LpProblem can fail to deliver a result
  - *assert problem.status == 1*
- Conclusion: Always test your solver !

Faster than PuLP

- CVXOPT - Python software for convex optimization
  - http://cvxopt.org/
  - Uses a different API than PuLP
  - Much better documentation
  - Up to 10-70x faster than PuLP
    https://scaron.info/blog/linear-programming-in-python-with-cvxopt.html
- CVXPY - Python-embedded modeling language for convex optimization problems
  - http://www.cvxpy.org/
- PICOS - Python Interface for conic optimization solvers
  - http://picos.zib.de/

Conference scheduling using Python

- PyCon UK: Conference Scheduler
  - https://github.com/PyconUK/ConferenceScheduler
  - Documentation: http://conference-scheduler.readthedocs.io/
  - Fairly new: only 2 months old
  - Uses PuLP, completely automated

- Alexander tried to use it for EuroPython 2017: failed due to exponential runtime

Conference scheduling using Python

- EuroPython 2017 Scheduler
  - Written by Alexander Hendorf
  - Doesn't use PuLP, but a similar model to the PyCon UK one
  - Works based on clustering + random shuffling + human touch

Thank you for your attention !

Beautiful is better than ugly.

Contact

eGenix.com Software, Skills and Services GmbH
Marc-André Lemburg
Pastor-Löh-Str. 48
D-40764 Langenfeld
Germany

eMail: mal@egenix.com
Phone: +49 211 9304112
Fax: +49 211 3005250
Web: http://www.egenix.com/