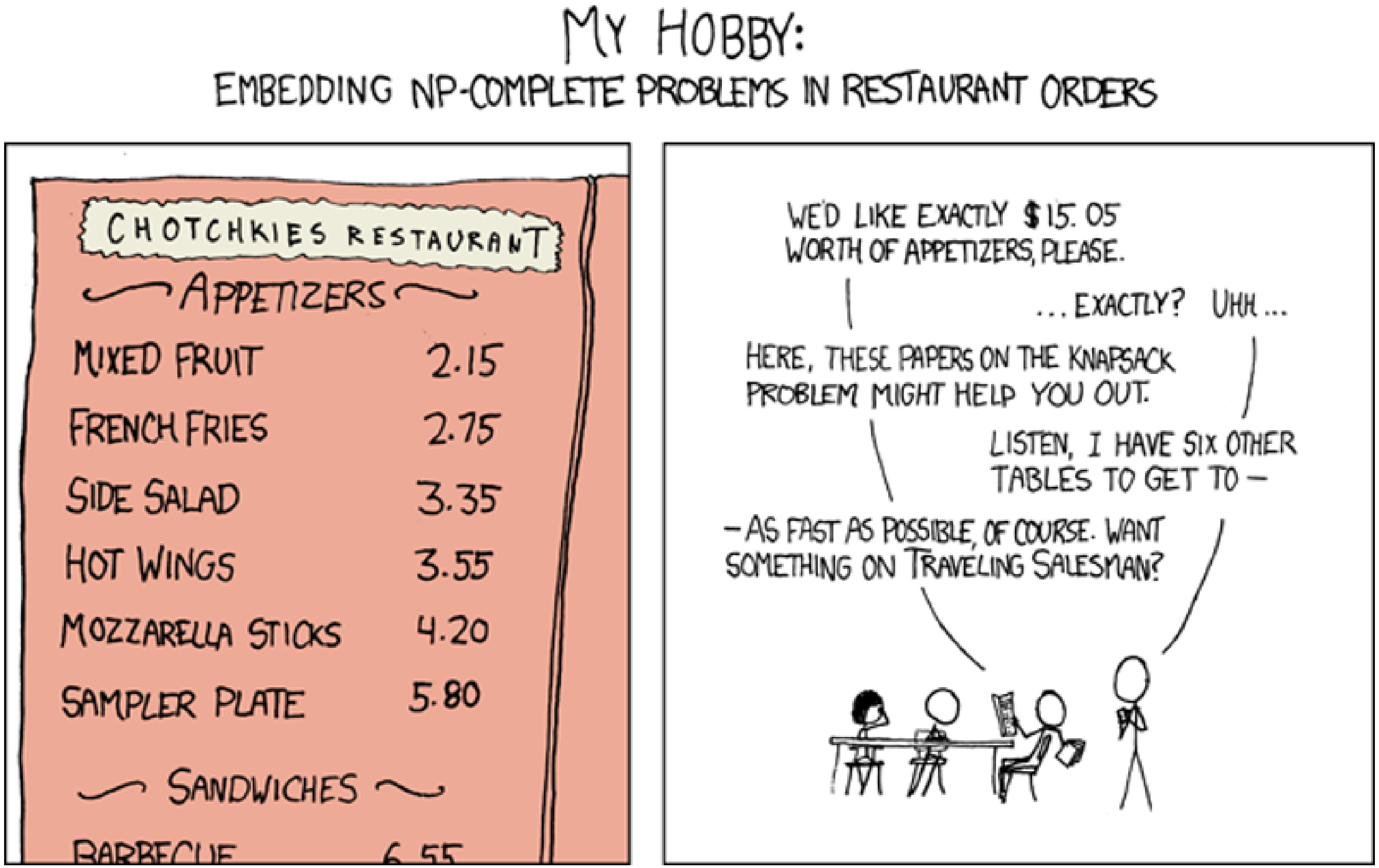
Introduction to Genetic Algorithms

Jan-Christoph Klie, Jana-Kira Schomber, Sebastian Muszytowski



source: http://xkcd.com/287/

Introduction to P-NP

- **P** and **NP** are classes of **decision problems**
- Usually answer is in the form $\{'yes', 'no'\}$
    - Example: "Is this number prime?"
    - Not a decision problem: "Primes between 42 and 1337"

P(olynomial) time problem

- **P** contains problems which compute **quickly**
- Informal: usual complexity in worst case polynomial

- Example
  - Problem: $\exists\ x \in S : even(x)$
  - Input: A sequence of n integers $n_1, n_2, \ldots$
  - Output: $true$ if any n_i is even, else $false$

N(ondeterministic)P(olynomial) time problem

- **NP** contains problems which can be verified in **polynomial time**
- Informal: usual complexity in worst case (brutefore): $\mathcal{O}(2^n)$

- Example
  - Problem: Sum of a subset of S is zero
  - Input: A set of integers S
  - Output: $true$ if the elements in a subset $A \subseteq S$ sums up to zero, else $false$

NP complete problem

- A problem **L** is **NP-complete if** $L \in NP$ **and** L **is NP-Hard**
- NP-hard problems are at least as hard as any problem in NP
- Not NP-complete problems are e.g. the halting problem

source: https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg

Darwinism

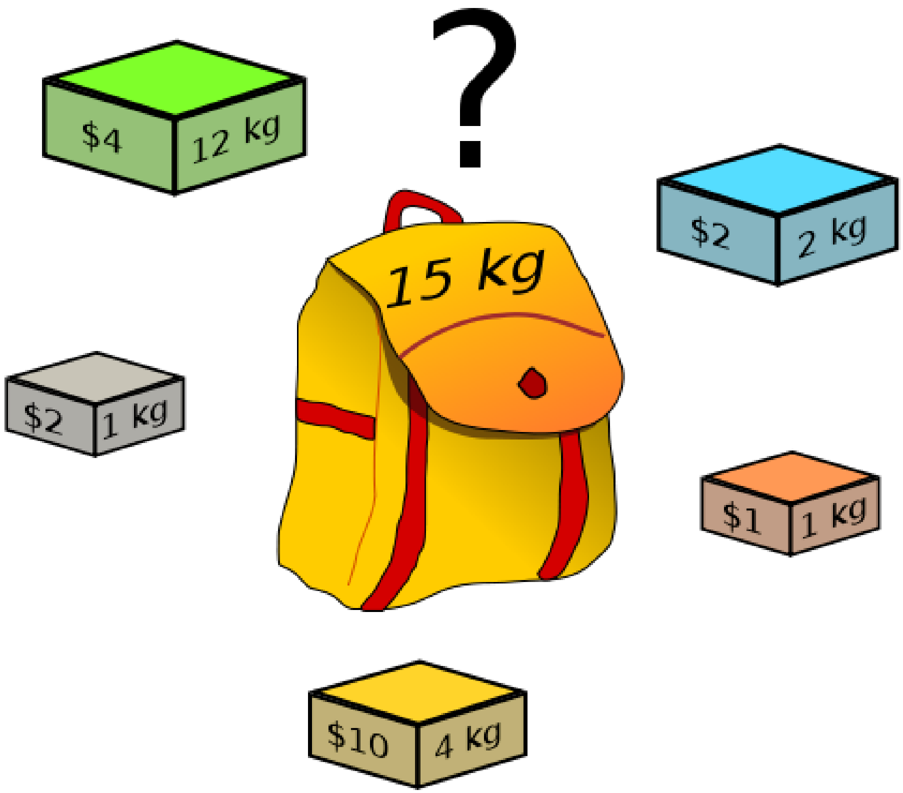
Survival of the Fittest

- "Natural role model" for Genetic Algorithms
- Random mutation of genes may enhance survival chances
- Next generation inherits mutated genes
- Repetition of the process leads to evolution

source: http://bioinformatics.oxfordjournals.org/content/24/1/1/F1.large.jpg

Optimization Problems

- Multiple solutions, where one or more are considered "optimal"
- One or more goals, by which to determine what is optional
- Constraints, which limit the solutions to feasible ones
- Among feasible solutions, the optimal must be found



source: http://en.wikipedia.org/wiki/File:Knapsack.svg

Genetic Algorithm and the Optimization Problem

MULTIPLE SOLUTIONS

Members make up a population

GOAL / GOALS

Condition/Conditions the fitness function should met

CONSTRAINTS

Condition/Conditions to determine which members die

Genetic Algorithm Recipe

- **Initialization:** Generation of random start population
- **Evaluation:** Calculate fitness for each member, check for termination critera
- **Generate a new population:**
    - **Selection:** Select fittest members for next generation
    - **Crossover:** Combine fittest members (parents) to generate new members (offspring)
    - **Mutation:** Vary the offspring to a certain degree to keep population a bit random to keep the game running

Genetic Knapsack Algorithm

The Foundation

CALCULATION BASIS

```
Item 1:    4€   12kg
Item 2:    2€    1kg
Item 3:    2€    2kg
Item 4:    1€    1kg
Item 5:   10€    4kg
```

SAMPLE MEMBER

```
   Item: 1 2 3 4 5
Bitmask: 1 0 0 1 1
```

Genetic Knapsack Algorithm

The Initial Random Population

RANDOM POPULATION AND EVALUATION

```
Member 1:   1 0 0 1 1  | 17kg, 15€
Member 2:   0 1 1 1 0  |  4kg,  5€
Member 3:   1 1 0 0 1  | 17kg, 16€
Member 4:   0 0 1 1 1  |  7kg, 13€
Member 5:   0 1 0 1 1  |  6kg, 13€
```

Genetic Knapsack Algorithm

Crossover

CROSSOVER

```
   Member 4:    0 0 1 1 1
 + Member 5:    0 1 0 1 1
========================
Offspring 1:    0 0 0 1 1
Offspring 2:    0 1 1 1 1
```

Genetic Knapsack Algorithm

Crossover

CROSSOVER

```
  Member 2:   0 1 1 1 0
+ Member 5:   0 1 0 1 1
========================
Offspring 3:   0 1 0 1 1
Offspring 4:   0 1 1 1 0
```

Genetic Knapsack Algorithm

Crossover

CROSSOVER

```
   Member 2:    0 1 1 1 0
 + Member 4:    0 0 1 1 1
========================
Offspring 5:    0 1 1 1 1
```

Genetic Knapsack Algorithm

Mutation

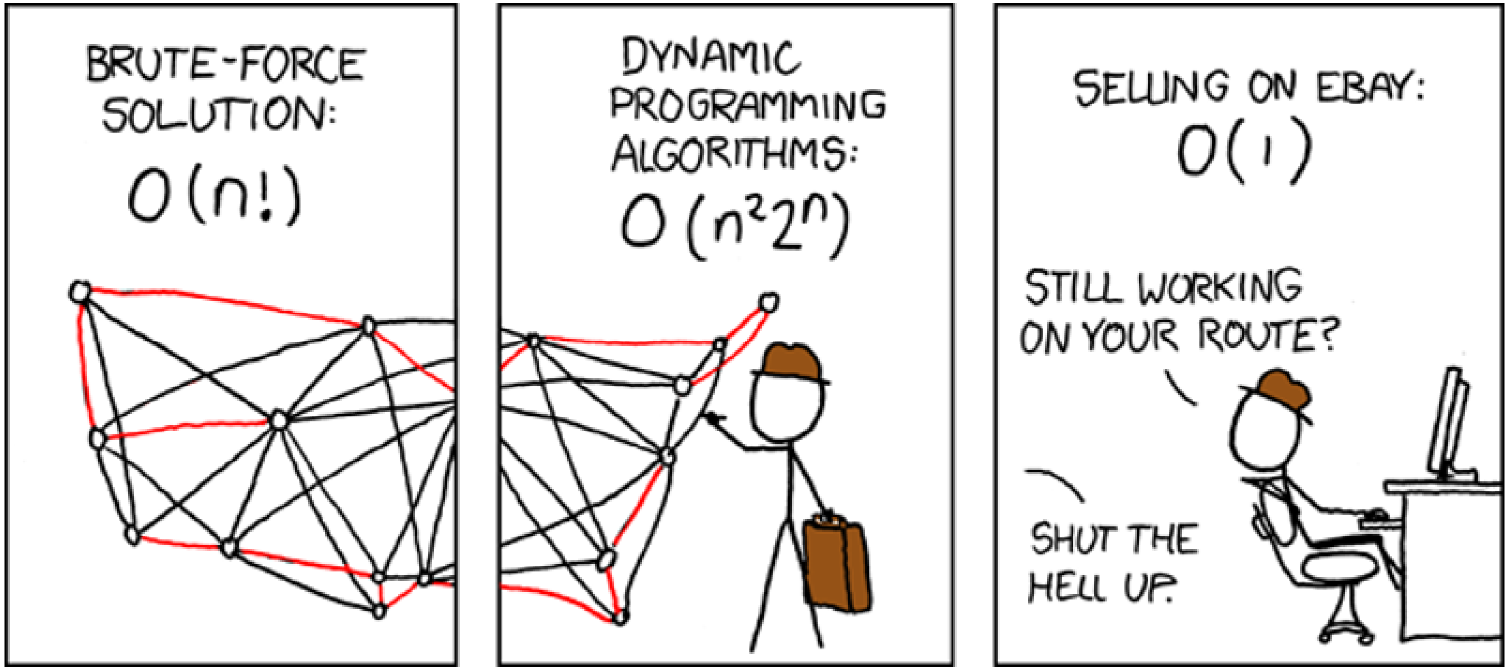
```
MUTATION

Offspring 1:   0 0 0 1 1  ->  1 0 0 1 1
Offspring 2:   0 1 1 1 1  ->  0 0 1 1 1
Offspring 3:   0 1 0 1 1  ->  0 1 1 1 1
Offspring 4:   0 1 1 1 0  ->  0 1 1 0 0
Offspring 5:   0 1 1 1 1  ->  0 1 1 1 0
```

Genetic Knapsack Algorithm

New Population

NEW POPULATION

```
Member 1:   1 0 0 1 1  | 17kg, 15€
Member 2:   0 0 1 1 1  |  7kg, 13€
Member 3:   0 1 1 1 1  |  8kg, 15€
Member 4:   0 1 1 0 0  |  3kg,  4€
Member 5:   0 1 1 1 0  |  4kg,  5€
```



source: http://xkcd.com/399/

The Traveling Salesman Problem

- Finding the shortest route within a list of cities
- NP complete problem with $\mathcal{O}(n!)$
- Optimal solution is hard to find, approximation is easier.

source: http://www.travelingsalesman.org/travelling_serial.html

Genetic Traveling Salesman

Definitions

POPULATION

set of solutions/routes (randomly initialized)

INDIVIDUAL

one single route

FITNESS

distance of the route

Genetic Travling Salesman

Basic Principle

PYTHON

```
# Create Population
initializePopulation()

# Work with Population
while Iteration < MaxGeneration:
   foreach Individual:
       calculateFitness()
   chromosomeSelection()
   chromosomeCrossing()
   chromosomeMutation()
   naturalSelection()
```

Genetic Travling Salesman

Fitness function

PYTHON

```
# Iterate over every population member
for j,pop in enumerate(population):
    cost[j]=0
    # Iterate over every "gene" of the member
    # and calculate the distance
    for z in range(cities):
        cost[j]=cost[j]+distances[pop[z],pop[z+1]]
```

Genetic Travling Salesman

Single Point Crossover

PYTHON

```
if np.random.rand() < crossing:
  cp=np.ceil(np.random.rand()*cities)
  for a in range(0, cp):
    child1[a] = parent2[a]
    child2[a] = parent1[a]
  for a in range(cp, cities):
    child1[a] = parent1[a]
    child[a] = parent2[a]
```

Genetic Travling Salesman

Mutation

PYTHON

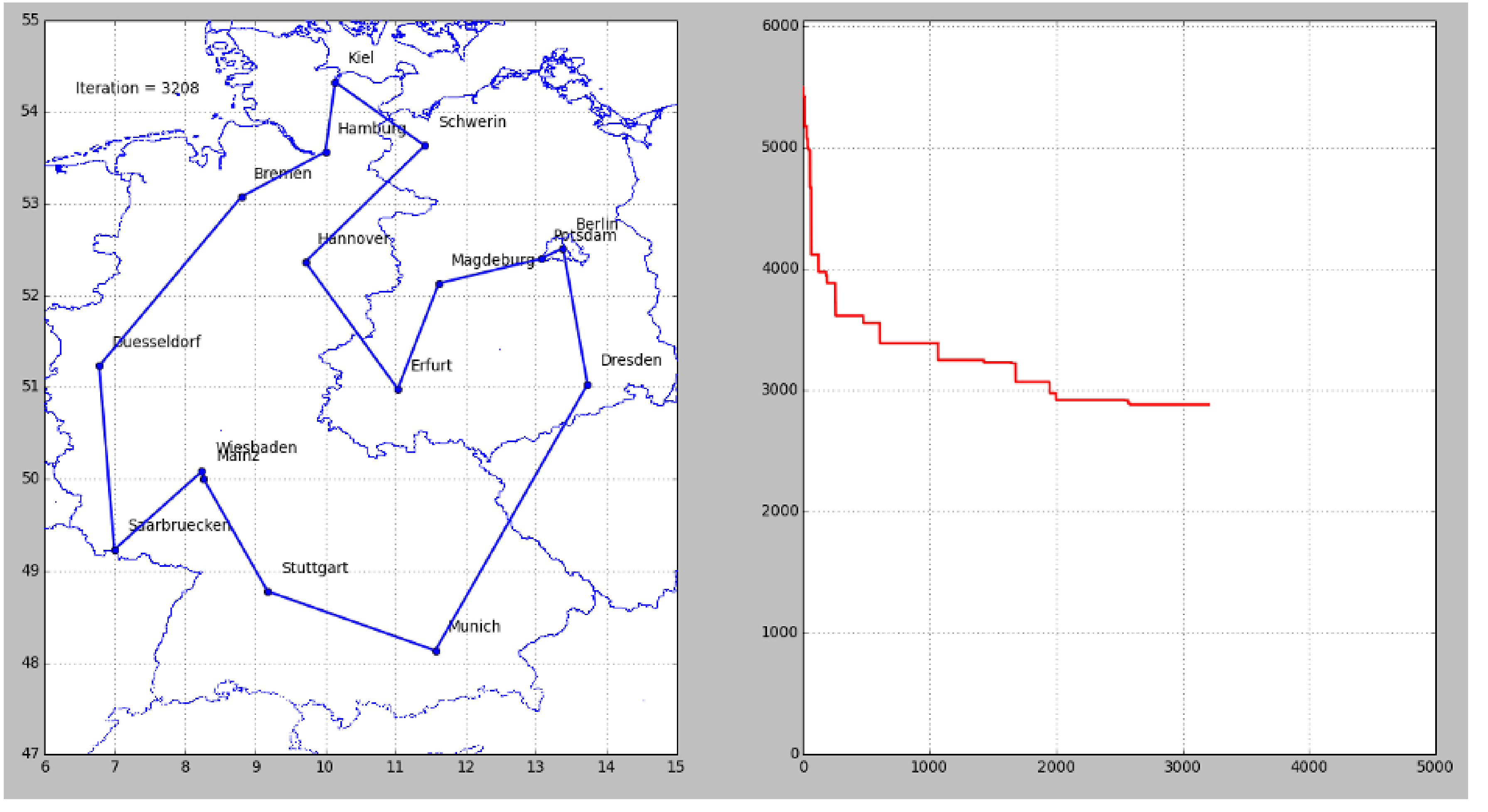
```
if np.random.rand()<mutation:
  mutInd=np.ceil(np.random.rand(2)*(cities-1))
  first=child1[mutInd[0]]
  second=child1[mutInd[1]]
  child1[mutInd[0]]=second
  child1[mutInd[1]]=first
  child1[-1]=child1[0] # last element and first element switch
```

Genetic Travling Salesman

Natural Selection

PYTHON

```python
for index in range(cities,0,-1):
    if sortedCost[index]>costChild1 and not replace1:
        if child1 not in sortedPopulation:
            sortedPopulation[index]=child1
            replace1=True
    elif sortedCost[index]>costChild2 and not replace2:
        if child2 not in sortedPopulation:
            sortedPopulation[index]=child2
            replace2=True
    if replace1 and replace2:
        break
```



source: screenshot of genetic algorithm implementation

Showtime!