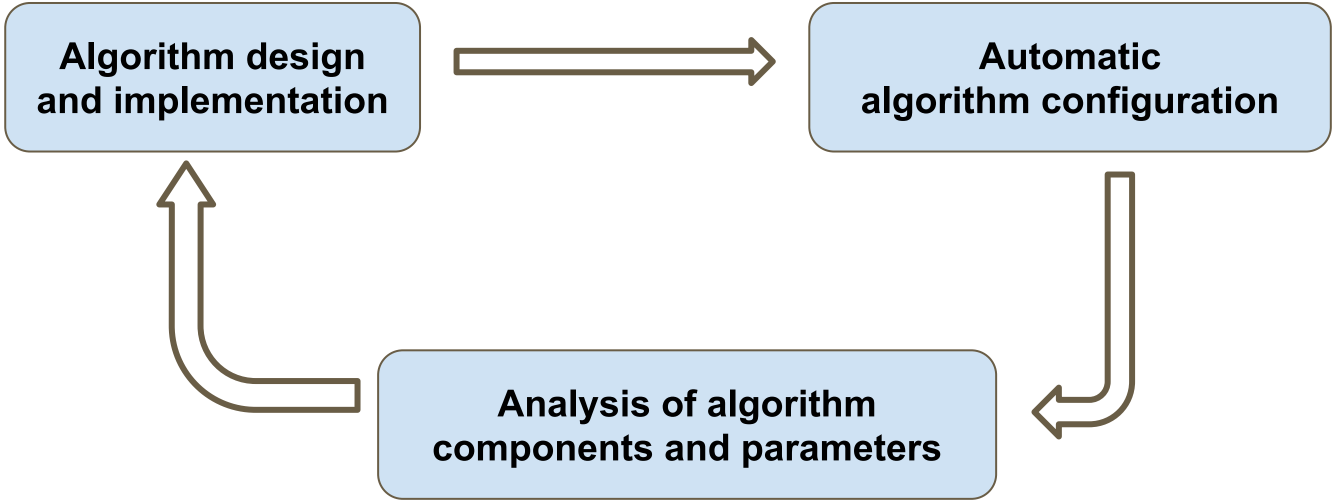
# Automatic algorithm configuration and analysis of algorithm parameters

Nguyen Dang

*School of Computer Science, University of St Andrews*

# Algorithm development process

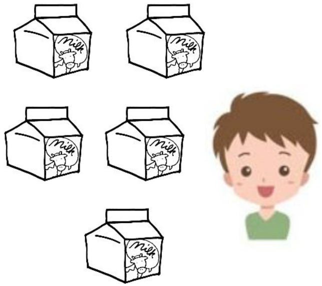


# Automatic algorithm configuration

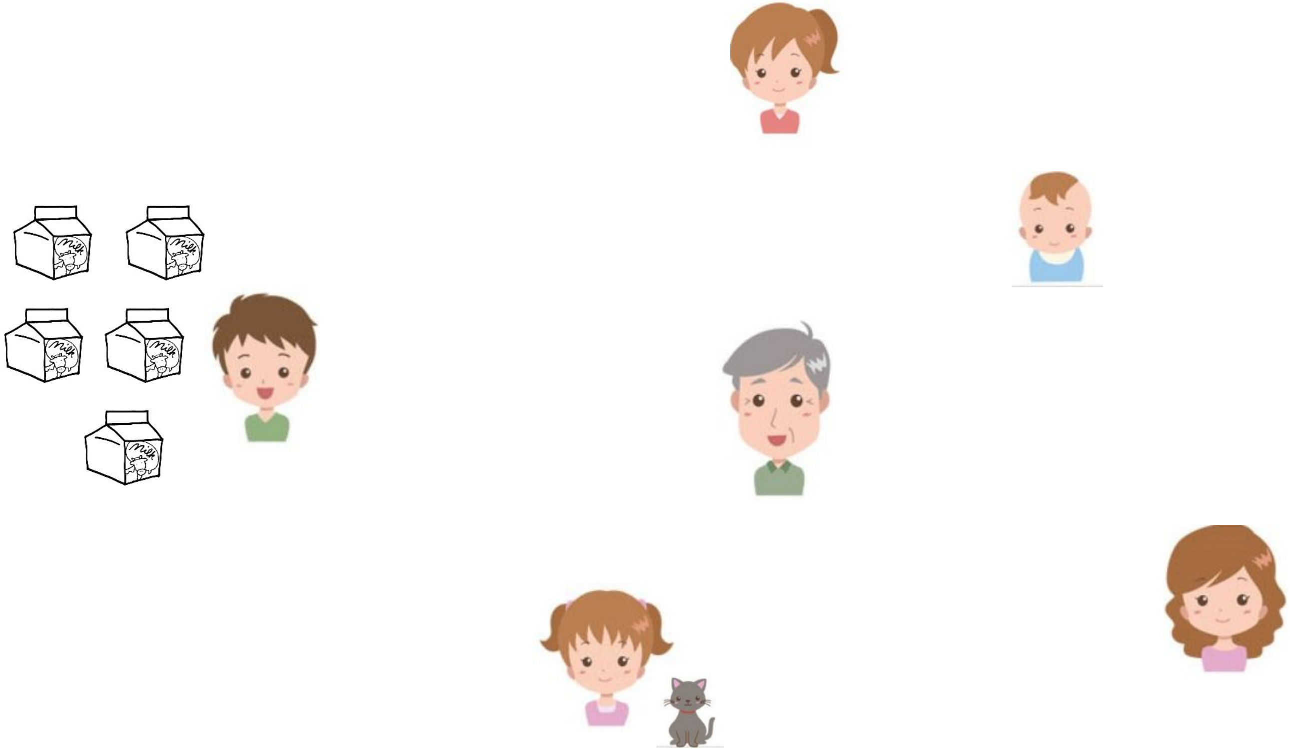
# Concepts

- Optimisation problem
- Problem instance and solution
- Parameterised algorithm
- Automatic algorithm configuration

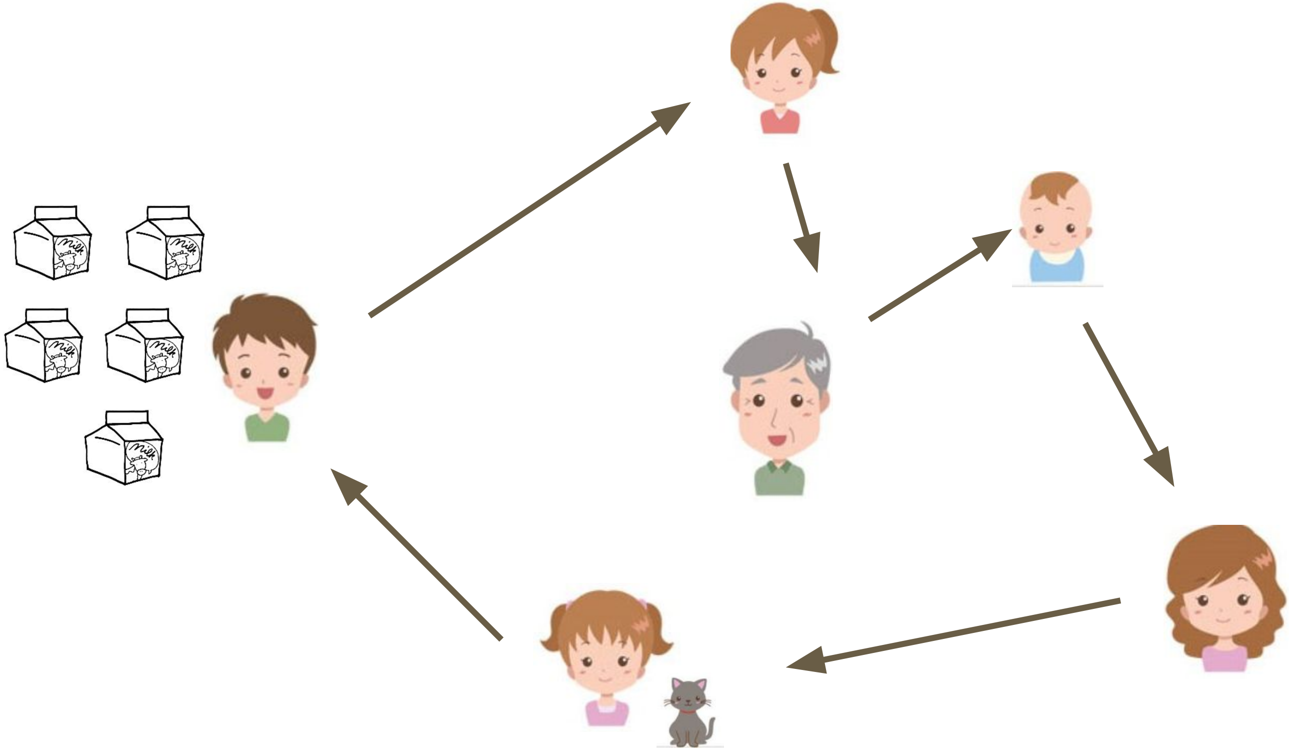
Hans has a milk delivery company.



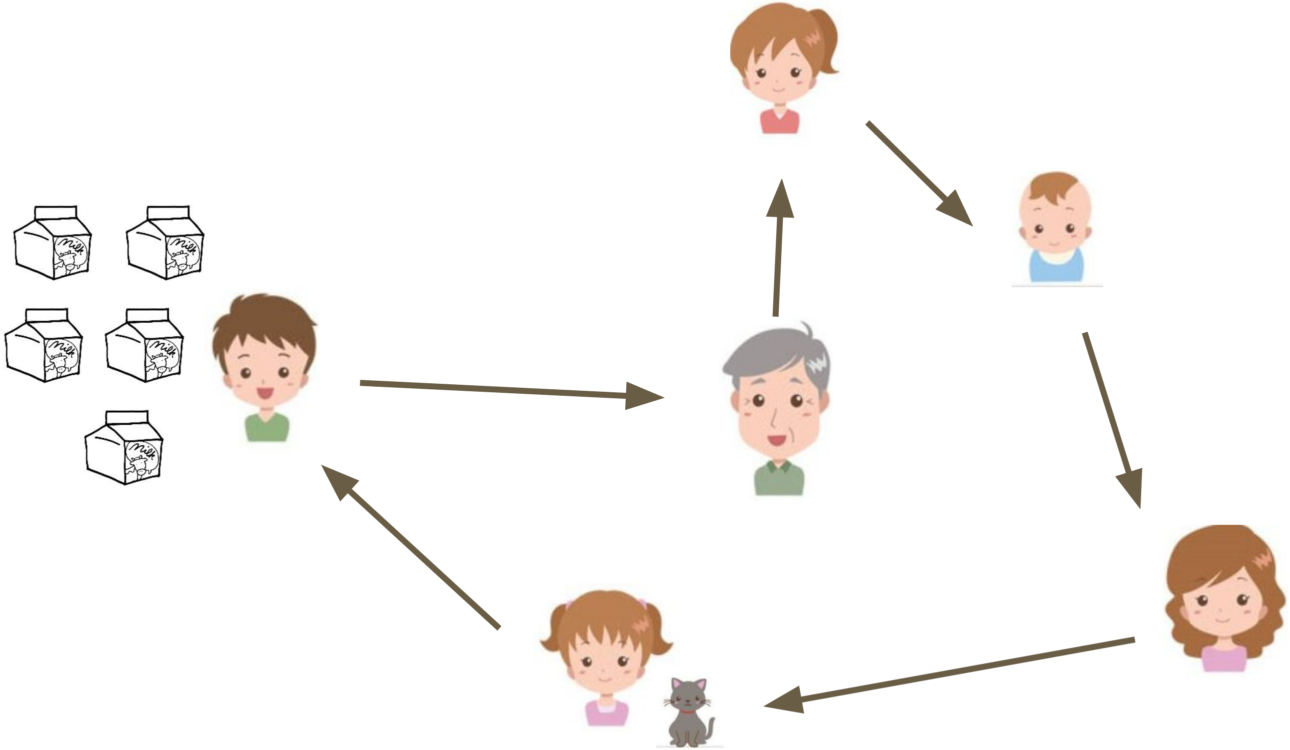
Everyday Hans receives orders and deliver milk to customers in town.



To save cost, Hans needs to find the ***shortest delivery route***.

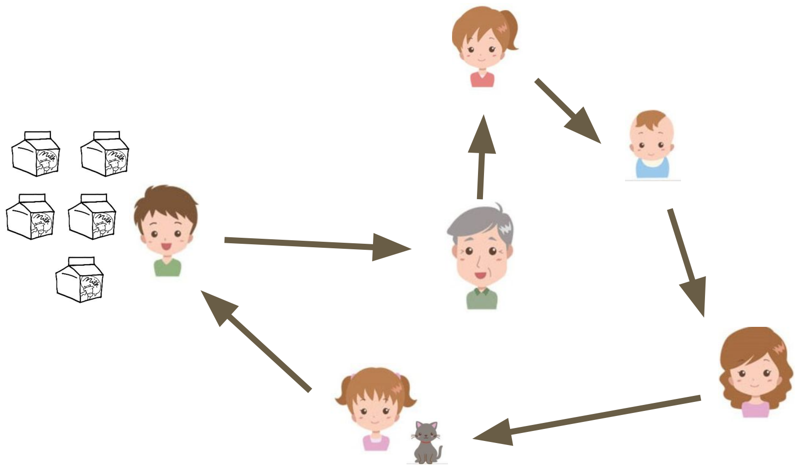


To save cost, Hans needs to find the ***shortest delivery route***.



# Optimisation problem

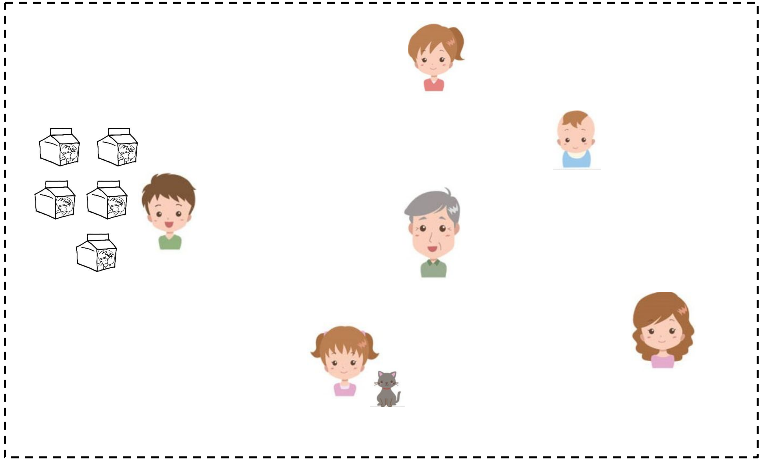
**Travelling Salesman Problem (TSP)**: given a list of places and the distances between each pair of places, what is ***the shortest possible route*** that visits each place exactly once and returns to the origin place?

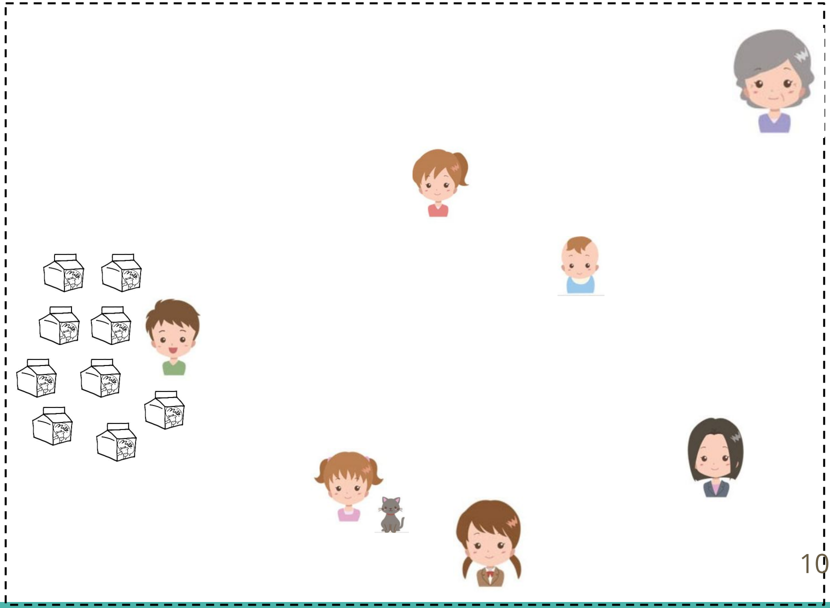


# Problem instance

**A problem instance** of the Travelling Salesman Problem (TSP) includes

- number of customers
- distance between every pair of customers



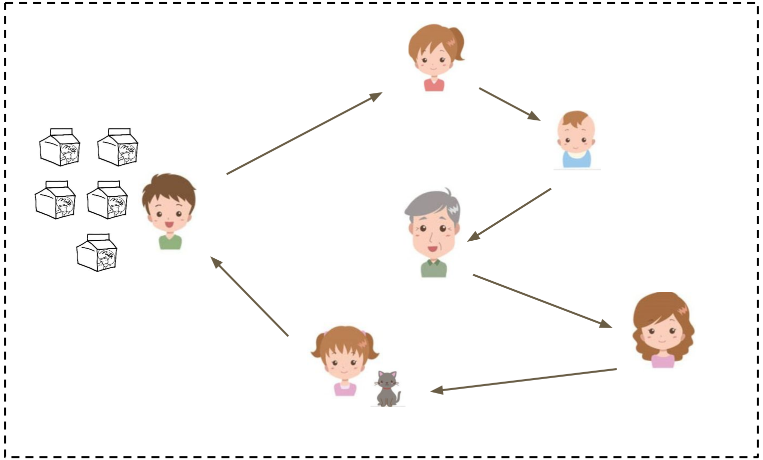


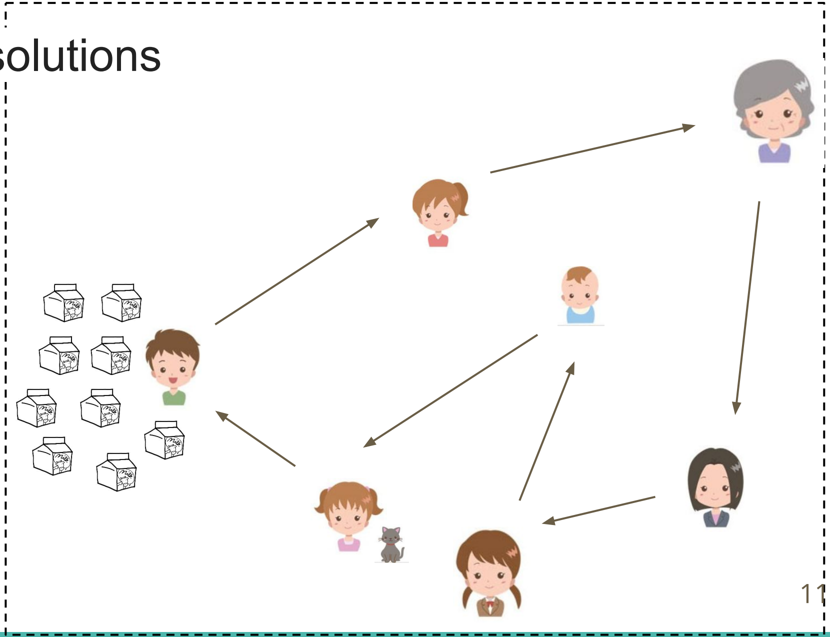
# Solution

**A solution** of a TSP instance: **a route**

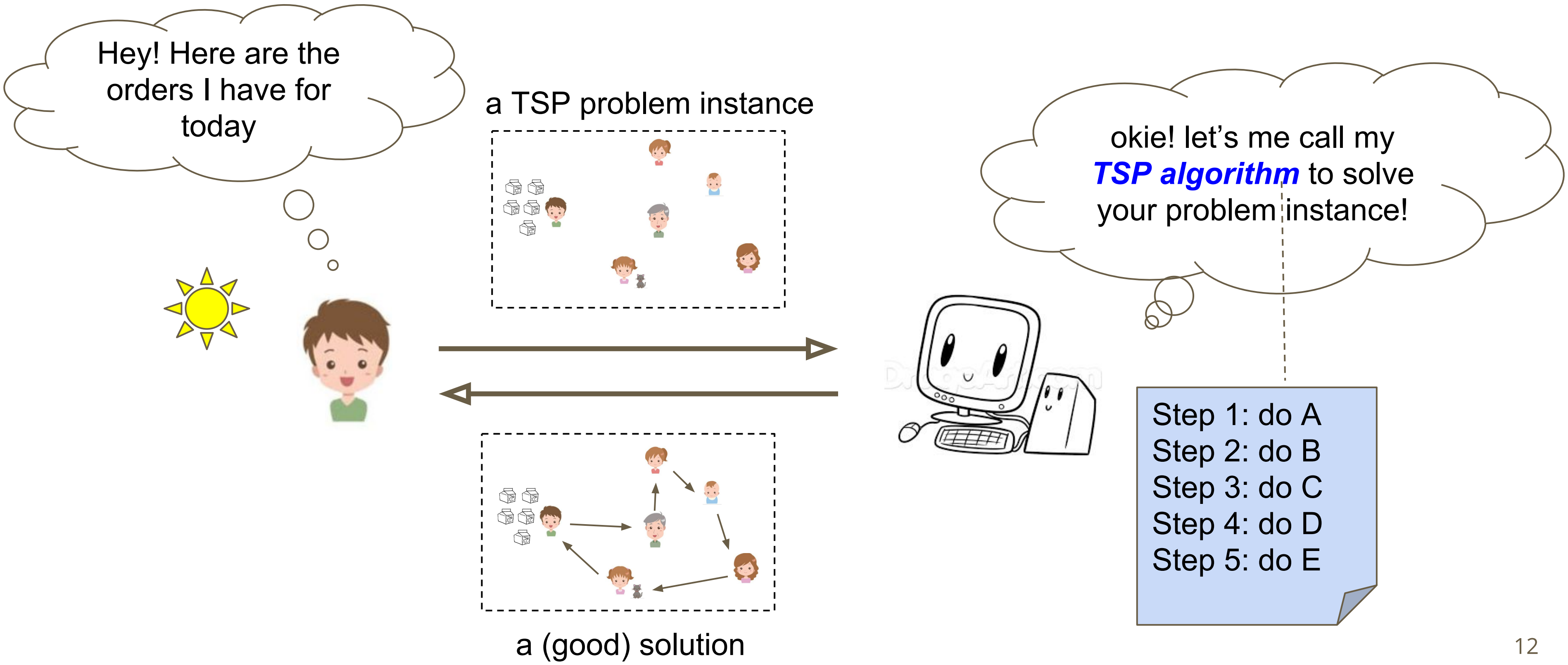
**Number of possible solutions**

- **n** customers: ***n!*** solutions
- **10** customers: ***3.628.800*** solutions
- **20** customers: ~ ***2.4 million milion million*** solutions





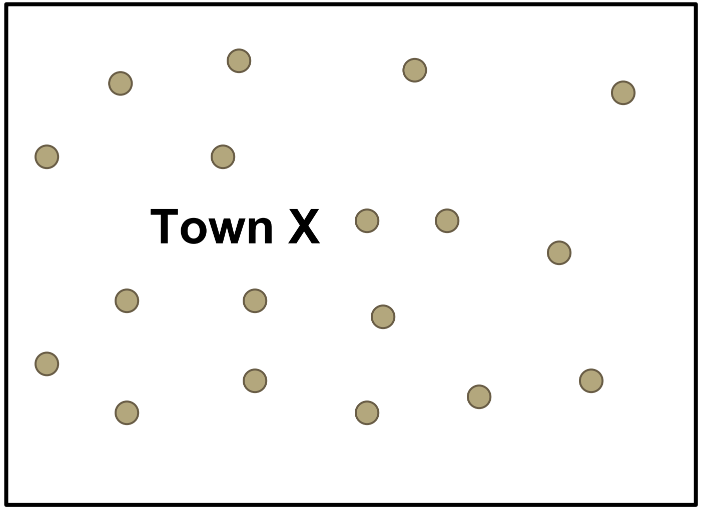
# Optimisation algorithm

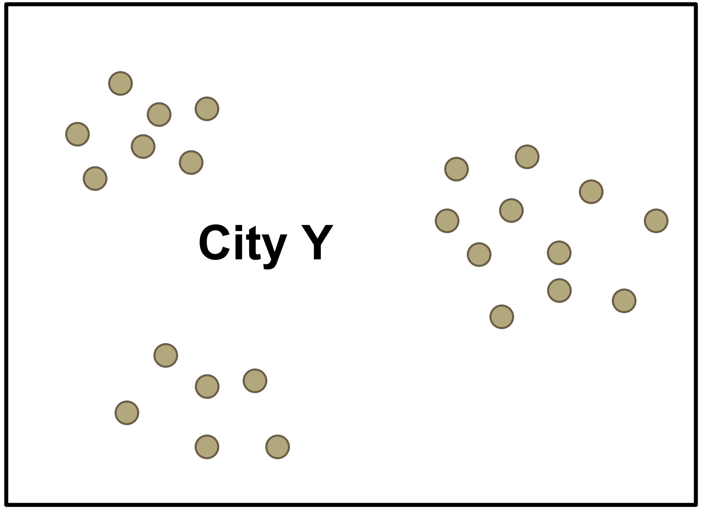


# Parameterised algorithms

Step 1: do A
Step 2: do B
Step 3: **do C**
Step 4: do D
Step 5: do E

Step 1: do A
Step 2: do B
Step 3: **do C’**
Step 4: do D
Step 5: do E



# Algorithm parameters and algorithm configuration

Step 1: do A
Step 2: do B
Step 3: **do *x***
Step 4: do D
Step 5: do E

***x*** ∈{C,C'}

***x*** is a ***categorical parameter*** of this TSP algorithm

Other ***parameter types***:

- ***ordinal***, e.g., *x* ∈ {A,B,C,D,E}
- ***integer***, e.g., *x* ∈ [1..5]
- ***continuous***, e.g., *x* ∈ [1.2, 2.5]

**Conditions** on parameters: y is activated only when x=='C'

***an algorithm configuration***: ***an instantiation of all algorithm's parameters***

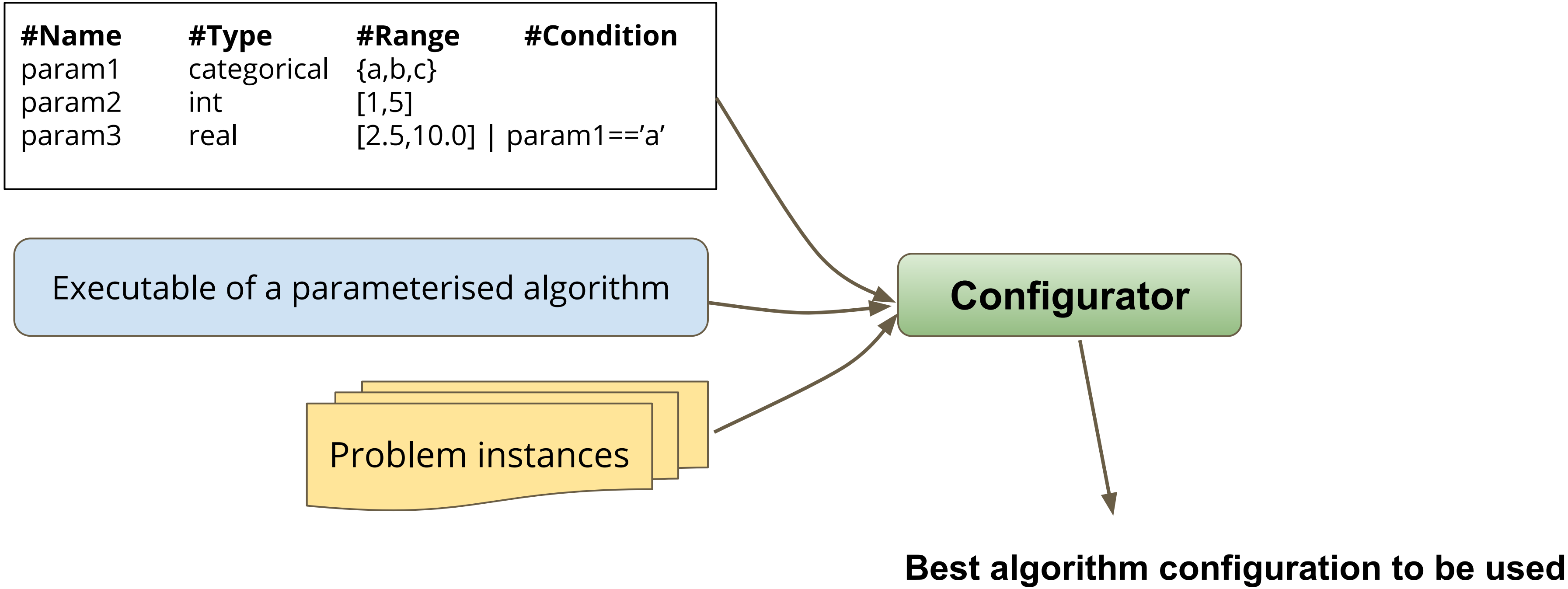
Step 1: do A
Step 2: do B
Step 3: **do C**
Step 4: do D
Step 5: do E

*configuration 1*

Step 1: do A
Step 2: do B
Step 3: **do C'**
Step 4: do D
Step 5: do E

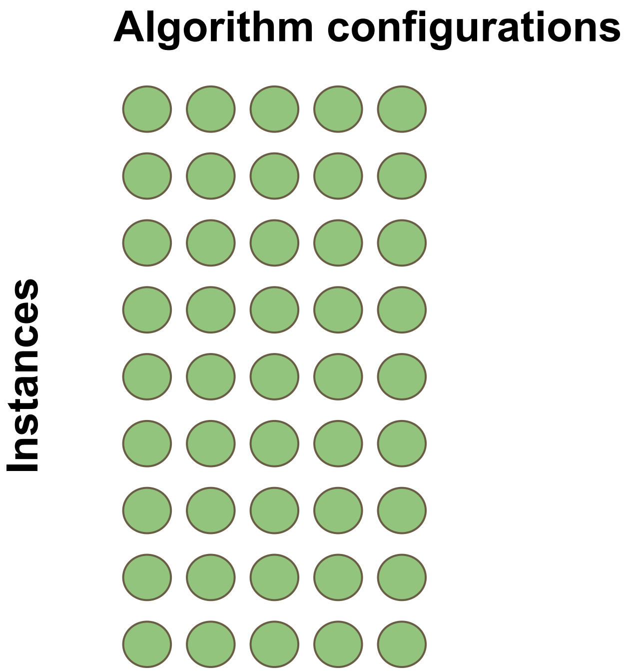
*configuration 2*

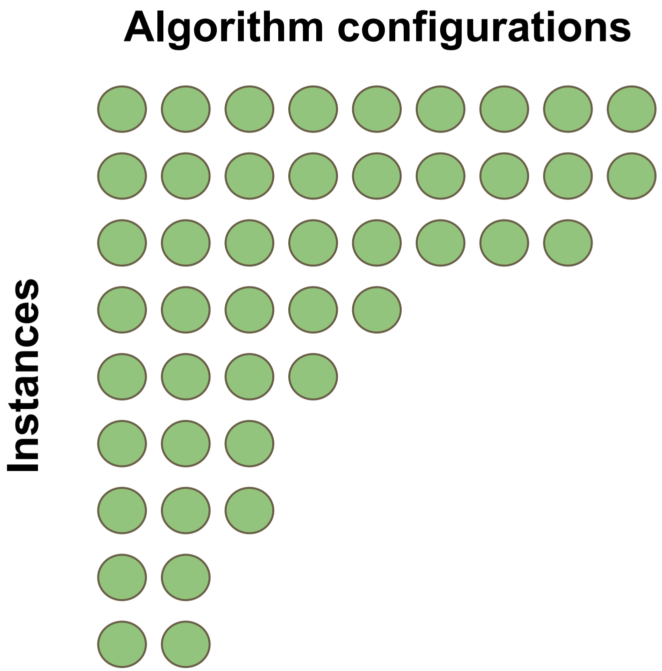
# Automatic algorithm configuration problem



# Racing technique

*Maron and Moore, 1994*




***Configuration budget is used efficiently!***

# Automatic algorithm configurators

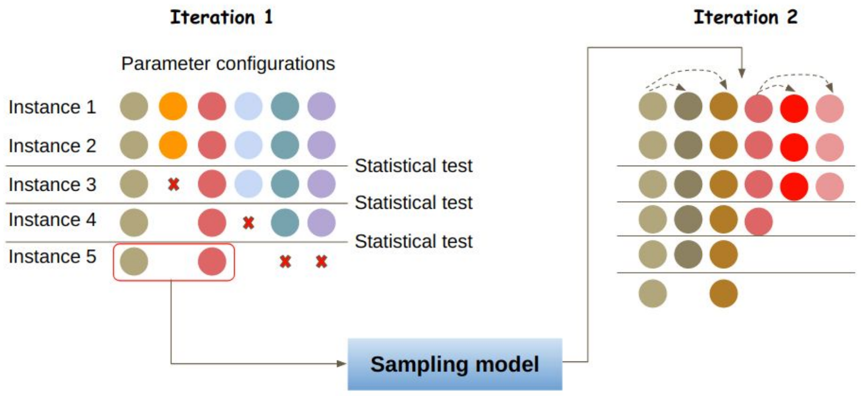
- **irace** (López-Ibánez et al 2011, 2016)
- **ParamILS** (Hutter 2009)
- **SMAC** (Hutter 2009)
- **GGA** and **GGA++** (Ansótegui et al 2009, 2015)

❖ well-packaged, easy to use
❖ support all types of parameters
❖ work on high-dimensional case studies

# Automatic algorithm configurators

**irace** *(López-Ibánez, Dubois-Lacoste, Pérez Cáceres, Birattari, and Stützle 2016)*

- http://iridia.ulb.ac.be/irace/
- available as an R-package on CRAN
- irace = iterated racing
  - a simple sampling model + racing
- parallelisation supported



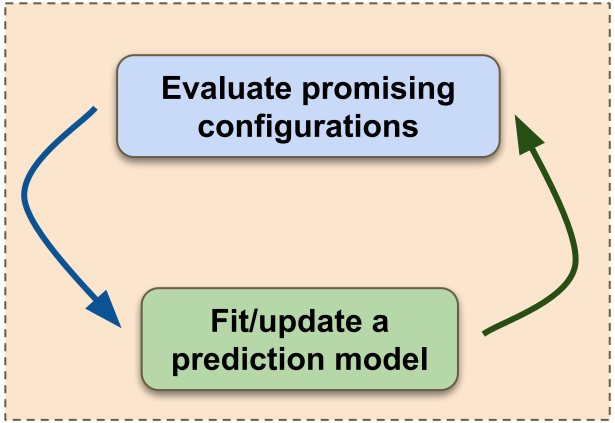
# Automatic algorithm configurators

**ParamILS** *(Hutter, Hoos, Leyton-Brown and Stützle 2009)*

- http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/
- written in Ruby
- ParamILS = Iterated Local Search in parameter configuration space
  - a sequential local search approach + racing between 2 configurations
- integer and continuous parameters must be discretised
- parallelisation not supported

  → multiple independent runs, take the best configuration in the end.

# Automatic algorithm configurators




**SMAC** *(Hutter, Hoos, Leyton-Brown and Stützle 2009)*

- http://www.cs.ubc.ca/labs/beta/Projects/SMAC/
- **SMAC** in Java, and **SMAC3** in Python
- SMAC = Sequential Model-based Algorithm Configuration
  - Bayesian Optimisation + racing between 2 configurations
- parallelisation: two options
  - Multiple independent runs
  - Multiple runs with shared model

# Automatic algorithm configurators

**GGA** *(Ansótegui, Sellmann, and Tierney 2009)*

**GGA++** *(Ansótegui, Malitsky, Samulowitz, Sellmann, and Tierney 2015)*

- GGA : https://bitbucket.org/gga_ac/dgga

  GGA++: https://bitbucket.org/mlindauer/aclib2
- written in C++
- GGA = Gender-based Genetic Algorithm
  - Genetic Algorithm + racing
- parallelisation supported

# Applications

- *Improve performance of state-of-the-art algorithms on several hard computational problems*: SAT, MIP, SMT, ASP, AI planning, TSP, BBOB, MOEA, robotics,...
  - CPLEX (ParamILS & SMAC, Hutter et al 2010): speed up with several order of magnitudes over default configuration, while CPLEX's tuning tool can't
- *Unified algorithm frameworks → automatic construction of algorithms*
  - Multi-objective evolutionary algorithm framework (irace, Bezerra et al 2014)
  - SATenstein (SMAC, Khudabukhsh et al 2009, 2016)
  - ...

# Applications

- Hyper-parameter optimisation in machine learning
    - **SMAC** has been successfully applied on a wide range of applications
        - SVM (Feurer et al 2015)
        - Neural networks (Domhan et 2015, Mendoza et al 2016)
        - Deep reinforcement learning (BOHB, Falkner et al 2018)
        - ...

      Used inside the **auto-sklearn** machine learning toolkit (Python)

https://www.automl.org/

# Analysis of algorithm parameters

# Analysis of algorithm parameters

- Output of an algorithm configuration procedure
  - the best algorithm configuration to be used
  - an algorithm performance dataset

  → can re-used to gain better insights into algorithm parameters

  using ***parameter analysis methods with prediction models***

# Analysis of algorithm parameters

Parameter analysis methods using prediction models

- forward selection (Hutter, Hoos & Leyton-Brown, 2013)
- fANOVA (Hutter, Hoos & Leyton-Brown, 2014)
- ablation analysis (Fawcett & Hoos, 2016; Biedenkapp, Lindauer & Eggensperger, 2017)

**Implemented in the Python tool PyImp**

**https://github.com/automl/ParameterImportance**

# Analysis of algorithm parameters

**Each method addresses a different analysis aspect**

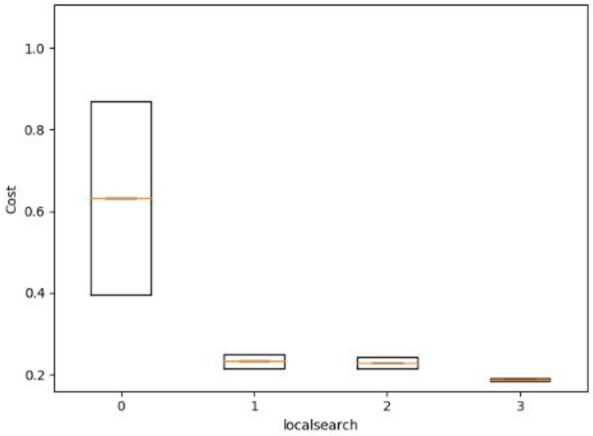
- *forward selection* (Hutter, Hoos & Leyton-Brown, 2013)
    - Identify a subset of key parameters
- *fANOVA* (Hutter, Hoos & Leyton-Brown, 2014)
    - Quantify the importance of every parameter and their pairwise interactions
- *ablation analysis* (Fawcett & Hoos, 2016; Biedenkapp, Lindauer & Eggensperger, 2017)
    - Quantify the importance of every parameter on the local path between two configurations

# Analysis of algorithm parameters

Example: Ant Colony Optimization algorithms for the Travelling Salesman Problem

```
All single-parameter effects: 87.63%
All pairwise interaction effects: 11.72%
localsearch: 77.85%
rho: 4.86%
... (remaining effects: < 4%)
```

**fANOVA analysis**




localsearch:

- 0: no local search
- 1: 2-opt (default)
- 2: 2.5-opt
- **3: 3-opt (irace's choice)**

# Summary

- Automatic algorithm configuration tools (SMAC, ParamILS, irace, GGA, GGA++)
  - easy to use
  - flexible, scalable
  - have a wide-range of applications
- Various analysis approaches after the configuration process
- Many more to try...

# Useful links

- https://www.automl.org/
- http://iridia.ulb.ac.be/irace/
- https://bitbucket.org/gga_ac/

# Thank you

# Appendix

# Automatic algorithm configurators

## irace - input

1. A parameter definition file
2. A set of problem instances (training/test)
3. An executable of the parameterised algorithm

`Input:` **<configuration_id> <instance_id> <random_seed> <instance_name> <param1> <param1_value> <param2> <param2_value>** …

`Output:` **<cost>,[<time>]**

4. Forbidden configurations (optional)
5. Default configurations (optional)
6. A scenario file: put everything together

# Automatic algorithm configurators

## irace - output

- Best configuration(s) to be used, and their performance on training/test set
- All detailed information is stored in an `.Rdata` file