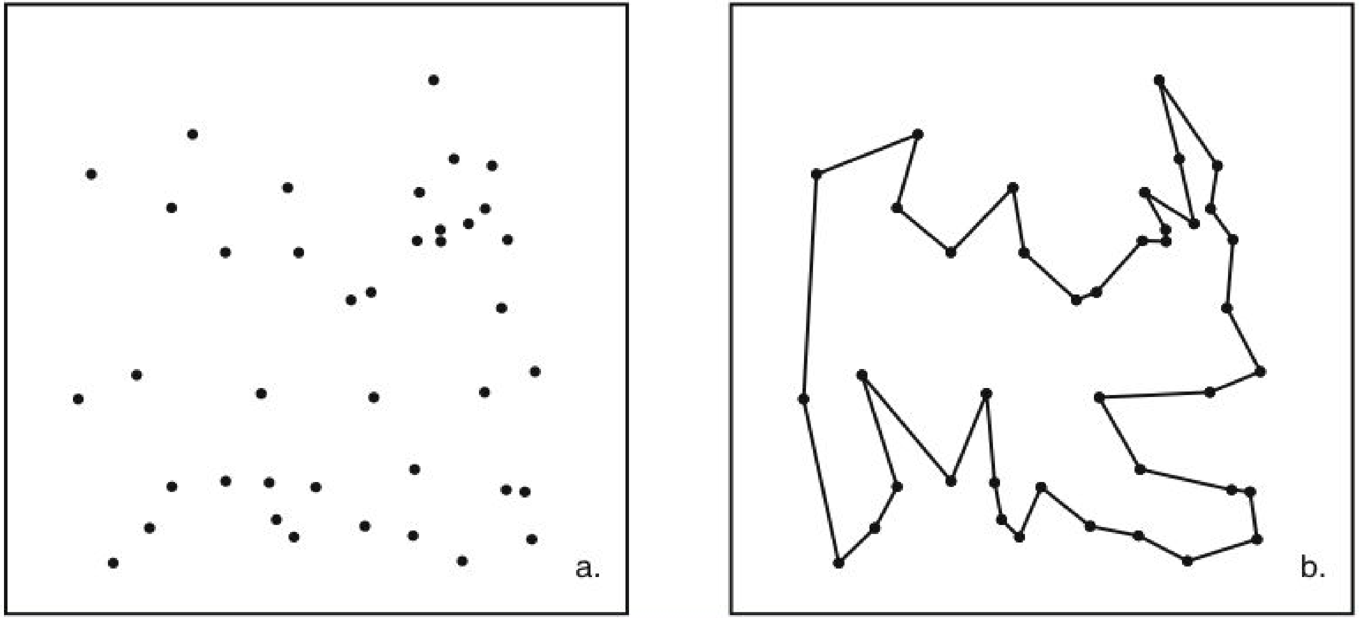
Automated parameter tuning with irace

Nguyen Thi Thanh Dang
nguyenthithanh.dang@kuleuven.be

(Combinatorial) Optimization problem

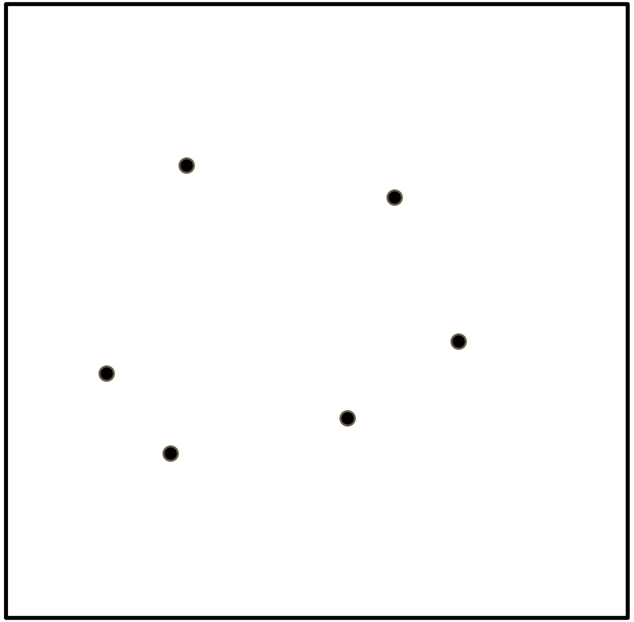
Travelling salesman problem: Given a list of cities and the distances between each pair of cities, what is ***the shortest possible route*** that visits each city exactly once and returns to the origin city?

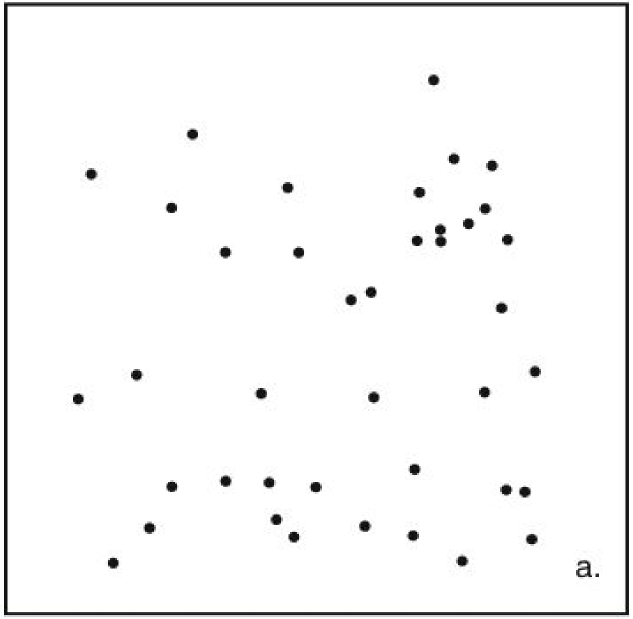


A problem instance (an instance)

Travelling salesman problem:

- The number of cities
- A distance matrix: distance between every pair of cities





A problem instance distribution

Example:

- #cities ~ U[1000,3000]
- x-coordinate ~ U[1,100]
- y-coordinate ~ U[1,100]

Algorithm parameters

Late Acceptance Hill Climber (Burke et al 2008): the length of the list

Simulated Annealing (Kirkpatrick et al 1983): initial temperature, cooling rate, number of iterations processed at each temperature

An algorithm configuration

An instantiation of the algorithm parameters

Algorithm parameter tuning

Given:

- A *training set* of problem instances I
- An algorithm $\mathcal{A}$ with a set of parameters $\mathcal{P}$
- A performance measure $f_{\mathcal{A},I}$

Find an algorithm configuration of $\mathcal{A}$ that optimize $f_{\mathcal{A},I}$

Some (off-line) automated parameter tuning tools:

- ***irace*** (iterated racing: López-Ibánez et al, 2011)
- ***SMAC*** (Sequential Model-based Algorithm Configuration: Hutter et al, 2011)

Comparison of two algorithm configurations

Given:

- A set of problem instances I

- Two algorithm configurations c_1 and c_2

- Performance (cost/time) of c_1 and c_2 on each instance of I

	configurations c_1	configurations c_2
instance i_1	45098	87648
instance i_2	654	434
instance i_3	7843	4873
instance i_4	342	43

solution cost
running time for reaching optimality
...

random seed

Comparison of two algorithm configurations

- Comparison of c_1 and c_2 on I
  + Mean/median of performance over all instances in I ***(SMAC)***
  + Statistical test ***(irace)***

	configurations c_1	configurations c_2
instance i_1	45098	87648
instance i_2	654	434
instance i_3	7843	4873
instance i_4	342	43

Comparison of two algorithm configurations

- Comparison of c_1 and c_2 on I
  - Mean/median of performance over all instances in I ***(SMAC)***
  - Statistical test ***(irace)***: based on ranks (default)

	configurations c_1	configurations c_2
instance i_1	1	2
instance i_2	2	1
instance i_3	2	1
instance i_4	2	1

(irace)

Comparison of two algorithm configurations using Wilcoxon-test:

- Apply (paired) Wilcoxon-test on performance data of c_1 and c_2
- If `p-value<(1-confident_level)` and `rank(c1)<rank(c2)`:

  conclude that c_1 is better than c_2

Comparison of more than two algorithm configurations using Friedman-test:

- Apply Friedman test on performance data of $(c_1, c_2, ..., c_k)$
- If `p-value<(1-confident_level)`
  - Let c_i be the configuration with the best ranking
  - For each $c_j \neq c_i$: apply post-hoc test on (c_i, c_j) to decide if c_i is significantly better than c_j or not
- Results: a list of ***alive (elite) configurations***

(see race.R: aux.friedman and aux2.friedman)

(irace)

At each iteration (race)

	c_1	c_2	c_3	c_4	c_5	c_6
i_1						
i_2						

(irace)

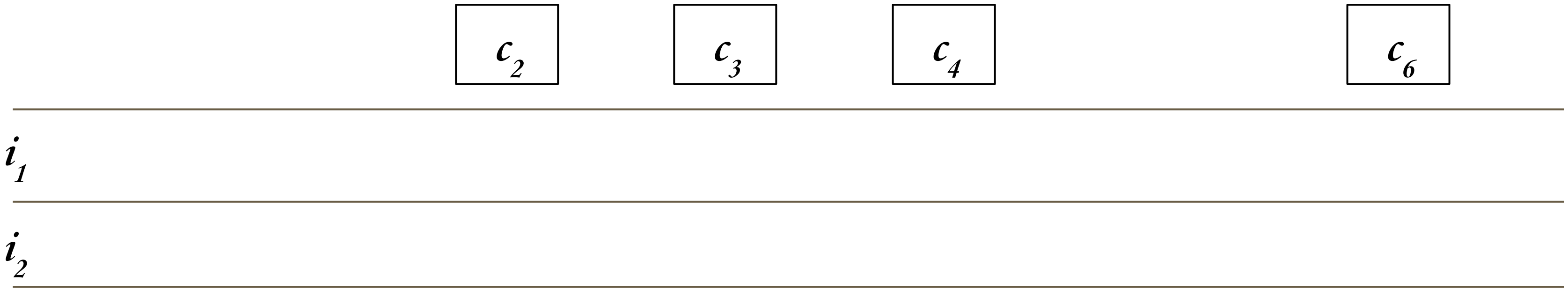
At each iteration (race)

	c_1	c_2	c_3	c_4	c_5	c_6
i_1						
i_2						

Apply statistical tests to eliminate significantly worse configurations

(irace)

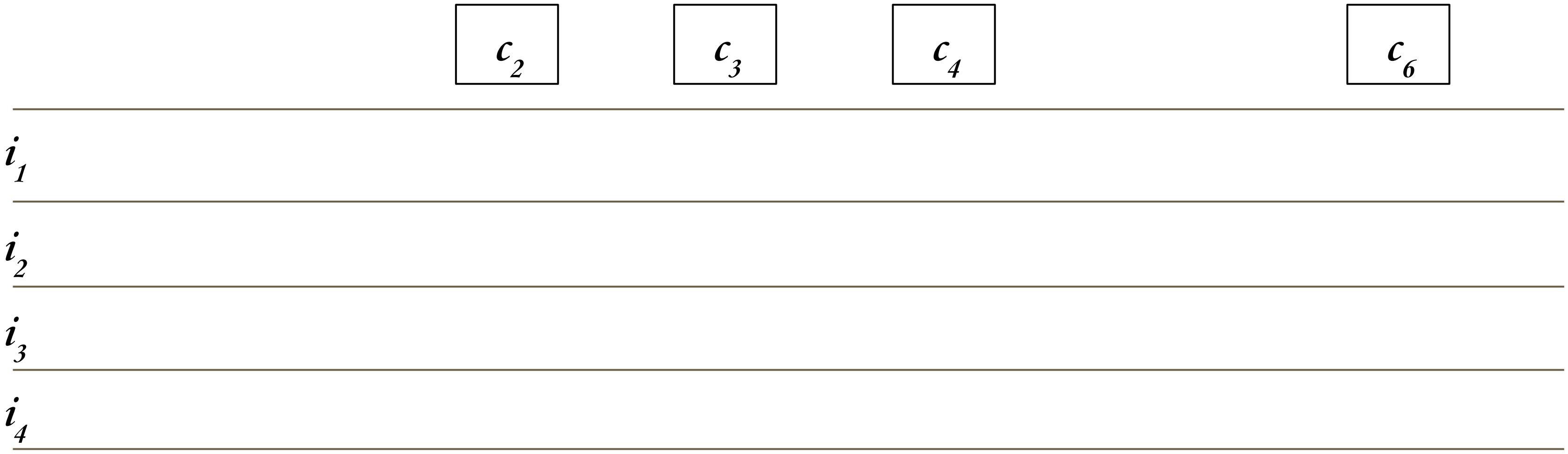
At each iteration (race)



Apply statistical tests to eliminate significantly worse configurations

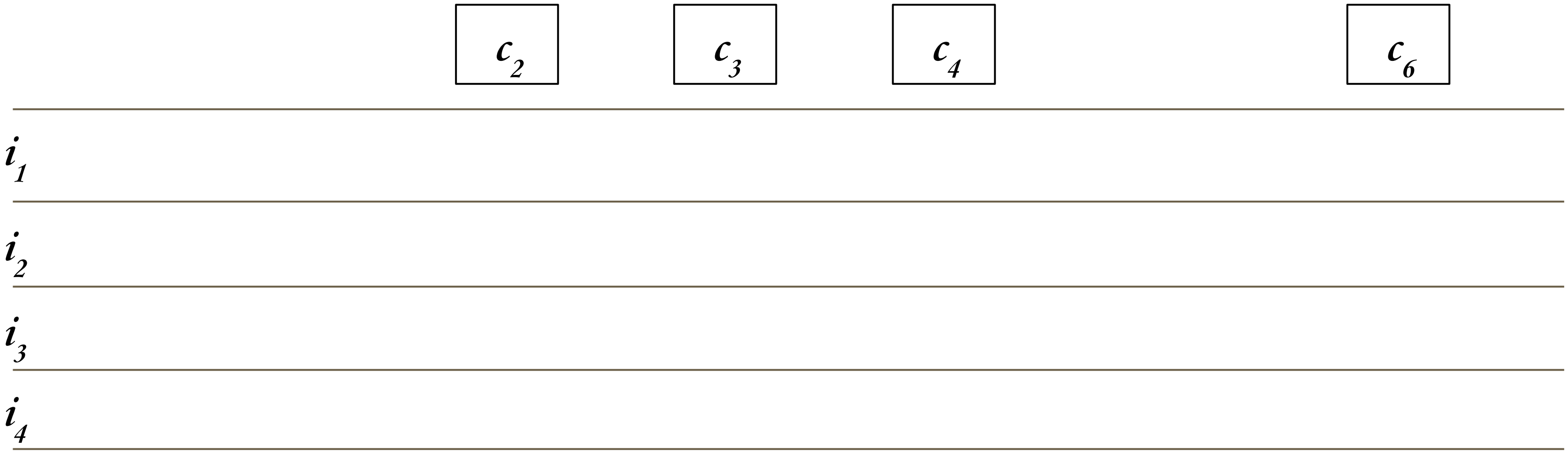
(irace)

At each iteration (race)



(irace)

At each iteration (race)



Results at the end of each iteration: a set of elite configurations

New configurations for the next iteration are sampled based on these elite configurations

irace

Output:

a (list of) good algorithm configuration(s)

irace

An R package

Website: http://iridia.ulb.ac.be/irace/

irace

Required input:

- Algorithm parameters description (*parameterFile*)
- A training instance set (*instanceFile and/or instanceDir*)
- Tuning budget (*maxExperiments*): how many algorithm runs?

Other input:

- A wrapper for calling the tuned algorithm on an instance (*hookRun*)
- A list of initial configurations (*candidatesFile*)
- Execution path (*execDir*)
- Parallelization (*parallel*, *mpi*)
- Debug level (*debugLevel*): information printed during the tuning
- Forbidden configurations

irace

Algorithm parameters description (*parameterFile*)

Information for each parameter:

- name
- switch
- type: *c (categorical), o (ordinal), i (integer), r (real)*
- values:
    - Categorical/ordinal parameter: a list of values
    - Integer/real parameters: lower bound and upper bound

Conditional parameter: is only activated according to specific values of some other parameter(s)

irace

A training instance set (*instanceFile* and/or *instanceDir*)

instanceDir: contains instance files

instanceFile: names of training instance files

irace

Tuning budget (*maxExperiments*): how many algorithm runs?

A run: an application of an algorithm configuration on an instance

A wrapper for calling the tuned algorithm on an instance (*hookRun*)

Arguments: <instance> <configuration_id> <switch_of_parameter_1> <value_of_parameter_1> <switch_of_parameter_2> <value_of_parameter_2> …

Output: a numeric performance value (printed to stdout)

irace

A list of initial configurations (*candidatesFile*)

Some configurations obtained from manual tuning/guessing/experience

Execution path (*execDir*)

Parallelization (*parallel*, *mpi*)

parallel: how many cores to use

mpi: normally for multiple nodes infrastructure, e.g., the VSC cluster (only activated when *parallel* > *1*)

irace

Debug level (*debugLevel*): information printed during the tuning

- 0: basic information only
- 1, 2: more advanced information

Forbidden configurations

irace

Try out an example:

Ant Colony Optimization for the Travelling Salesman Problem

- In irace folder: check the example in inst/examples/acotsp/
- You will also need to download the training instance files and the ACOTSP software (the tuned algorithm)