11 Parameter Tuning and Control

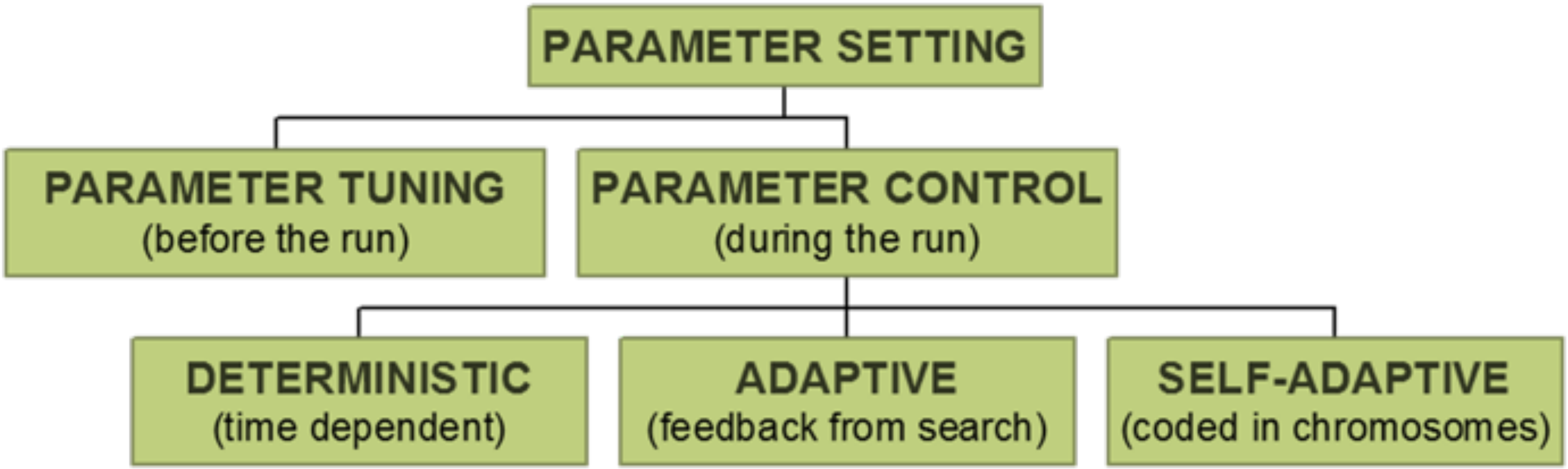
Parameters and Parameter Tuning

- we're going to take a look at these things:
- history
- taxonomy
- parameter tuning vs parameter control
- EA calibration
- parameter tuning
    - testing
    - effort
    - recommendations

Brief Historical Account

- 1970/80s "GA is a robust method"
- 1970s + ESs self-adapt mutation stepsize σ
- 1986 meta-GA for optimizing GA parameters
- 1990s EP adopts self-adaptation of σ as 'standard'
- 1990s some papers on changing parameters on the-fly
- 1999 Eiben-Michalewicz-Hinterding paper proposes clear taxonomy & terminology

Taxonomy


PARAMETER SETTING
PARAMETER TUNING
(before the run)
PARAMETER CONTROL
(during the run)
DETERMINISTIC
(time dependent)
ADAPTIVE
(feedback from search)
SELF-ADAPTIVE
(coded in chromosomes)

Parameter Tuning

- testing and comparing different values before the ‘real’ run
- problems:
    - users mistakes in settings can be sources of errors or sub-optimal performance
    - costs a lot of time
    - parameters interact
        - so exhaustive search is not practicable
    - good values may become bad during the run

Parameter Control

- setting values on-line, during the actual run
- for example:
    - predetermined time-varying schedule p = p(t)
    - using (heuristic) feedback from the search process
    - encoding parameters in chromosomes and rely on natural selection
- problems:
    - finding optimal p is hard, finding optimal p(t) is harder
    - still user-defined feedback mechanism, how to optimise?
    - when would natural selection work for algorithm parameters?

Notes on Parameter Control

- parameter control offers the possibility to use appropriate values in various stages of the search
- adaptive and self-adaptive control can liberate users from tuning
    - so reduces need for EA expertise for a new application
- assumption: control heuristic is less parameter-sensitive than the EA

but...

- state-of-the-art is a mess
- literature is a potpourri:
    - no generic knowledge
    - no principled approaches to developing control heuristics (deterministic or adaptive)
    - no solid testing methodology

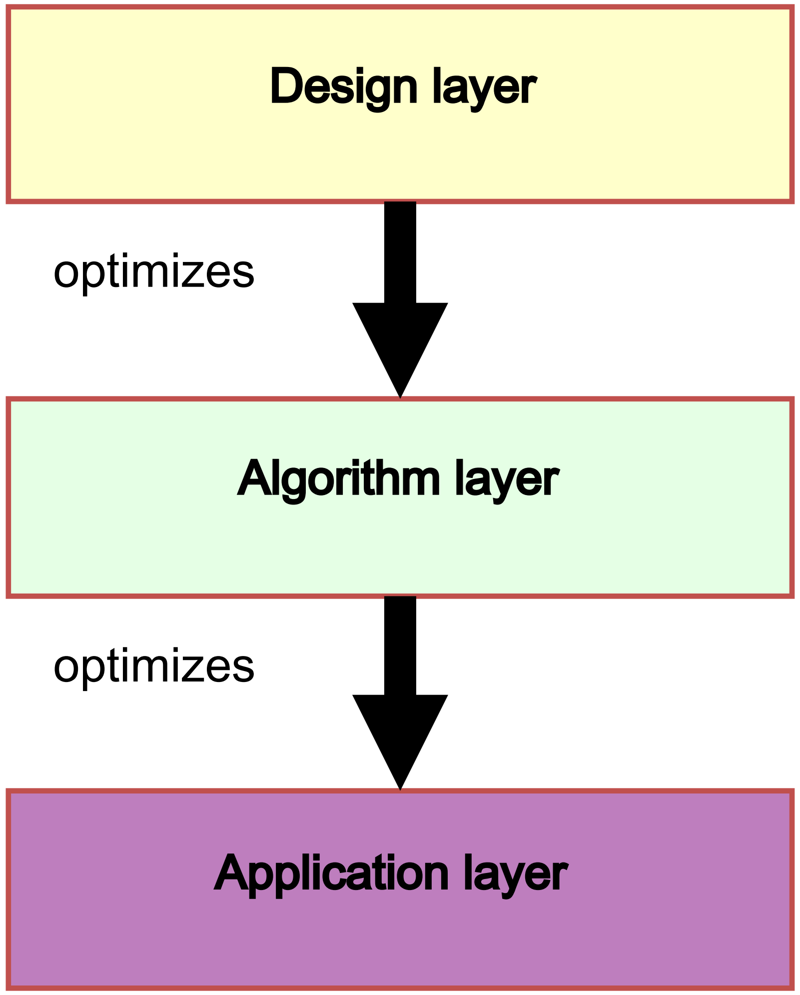
Historical Account (continued)

- in the last decade:
- more & more work on parameter control
  - traditional parameters: mutation and xover
  - non-traditional parameters: selection and population size
- not much work on parameter tuning, i.e.,
  - nobody reports on tuning efforts behind their EA published
  - a handful of papers on tuning methods / algorithms

Designing Evolutionary Algorithms

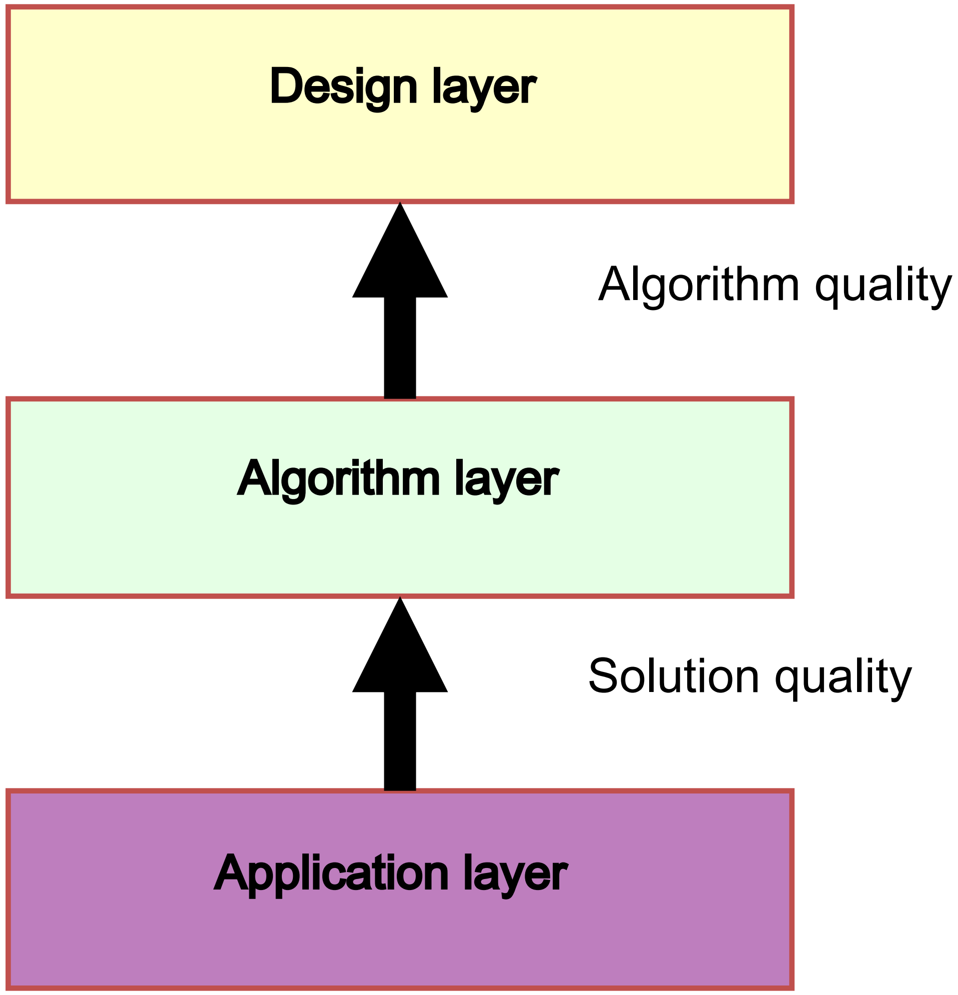
- we can think of EA as having 3 layers:
  - application
  - algorithm
  - design
- these layers interact in two fundamental ways

Control F;ow of EA Calibration / Design



the entity on a given layer optimises the entity on the layer below

Information Flow of EA Calibration / Design

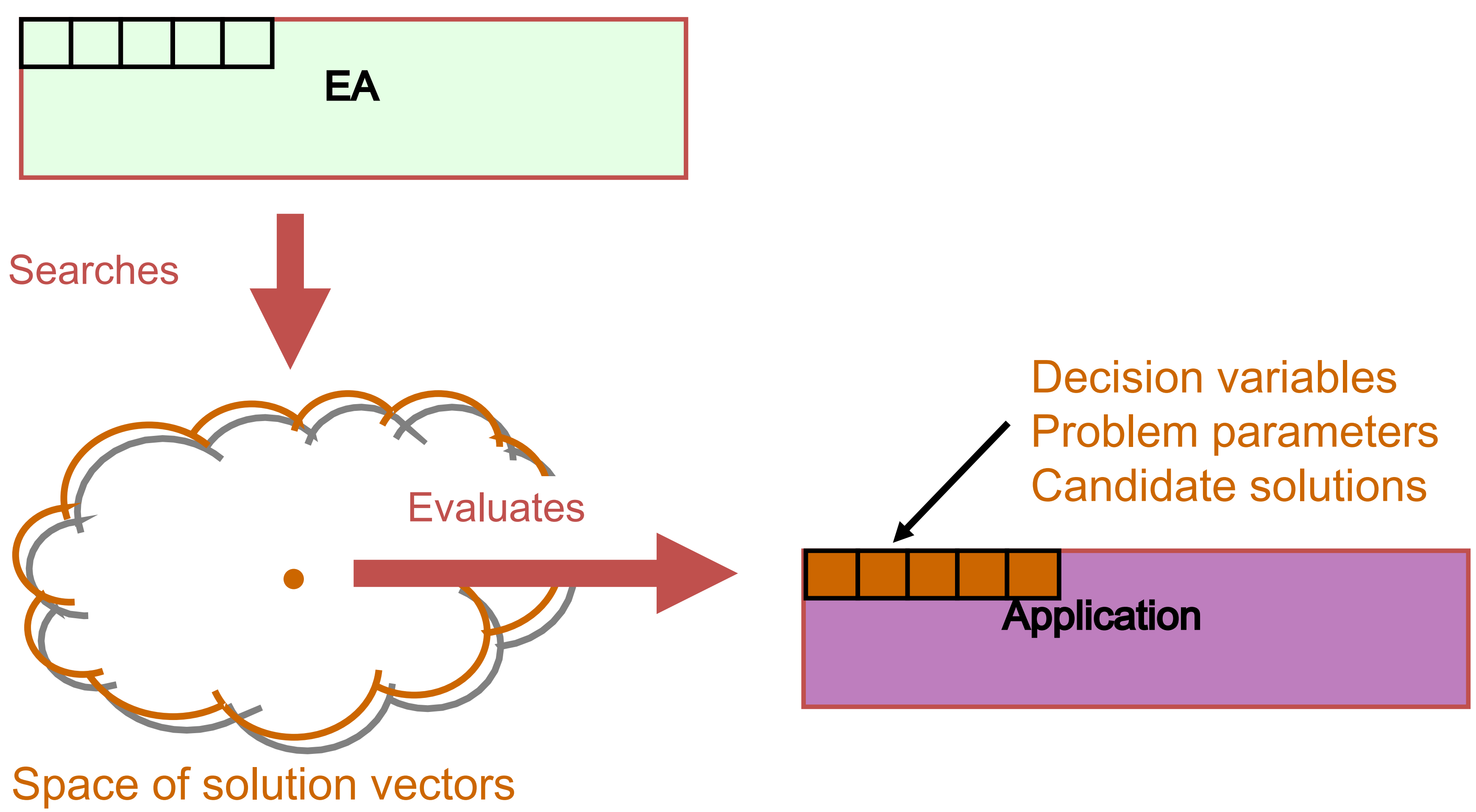


the entity on a given layer provides information to the entity on the layer above

Lower Level of EA Calibration / Design

problem solving

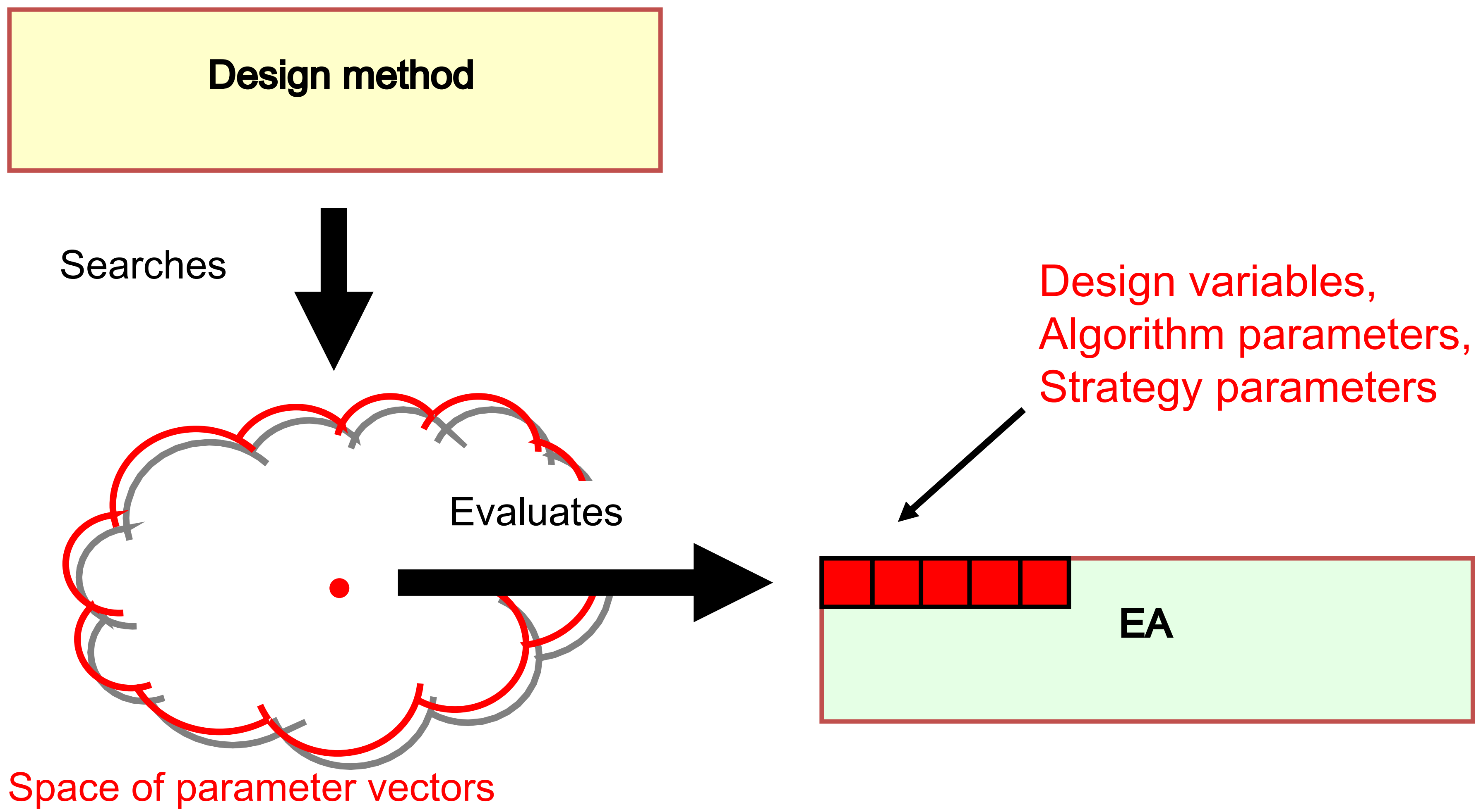
- an EA instance at the algorithm layer is trying to find an optimal solution for the given problem instance at the application layer



Upper Level of EA Calibration / Design

design method

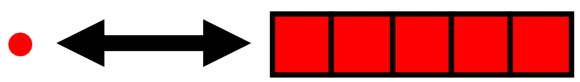
- the intuition and heuristics of a user or automated design strategy
- strategy is trying to find optimal parameter values for the given EA at the algorithm layer
- the quality of a parameter vector is based on the performance of the EA instance using those values



Testing Utility

- at the application layer we evaluate fitness (of an EA instance)
- at the algorithm layer we test utility (of the parameter vector)
- so the utility landscape is an abstract landscape where the locations are the parameter vectors of an EA and the height reflects utility
- this is similar to a fitness landscape, but there are differences:
  - for most problems, fitness values are deterministic
    - but utility values are always stochastic
    - because they reflect the performance of an EA, which is a stochastic search method
    - so maximum utility needs to be defined in some statistical sense
  - how 'good' the parameters are depends on context
    - do we just want to find the best solution for a single problem instance?
    - or be able to repeatedly solve instances of the same problem type?

Utility Landscape

- all parameters together span a search space
- which forms a landscape
- one point ↔ one EA instance 
- height of point = performance of EA instance
    - on a given problem
- this landscape is unlikely to be trivial
- ff there is some structure in the utility landscape, then we can do better than random or exhaustive search

Vocabulary For Problem Solving and Algorithm Design

	LOWER PART	UPPER PART
METHOD	EA	Tuner
SEARCH SPACE	Solution vectors	Parameter vectors
QUALITY	Fitness	Utility
ASSESSMENT	Evaluation	Test

Algorithm Quality: Performance

- there are two basic performance measures for EAs:
  - the fitness function, which measures solution quality
  - algorithm speed, such as number of fitness evaluations, CPU cycles, wall-clock time
- three basic combinations of these can be used to define the algorithm performance of a single run:
  - fix time and measure quality
  - fix quality and measure time
  - fix both and measure completion
- but a good estimation of performance always requires multiple runs on the same problem with the same parameter values
- and some aggregation of the measures used for each run

Common Performance Metrics

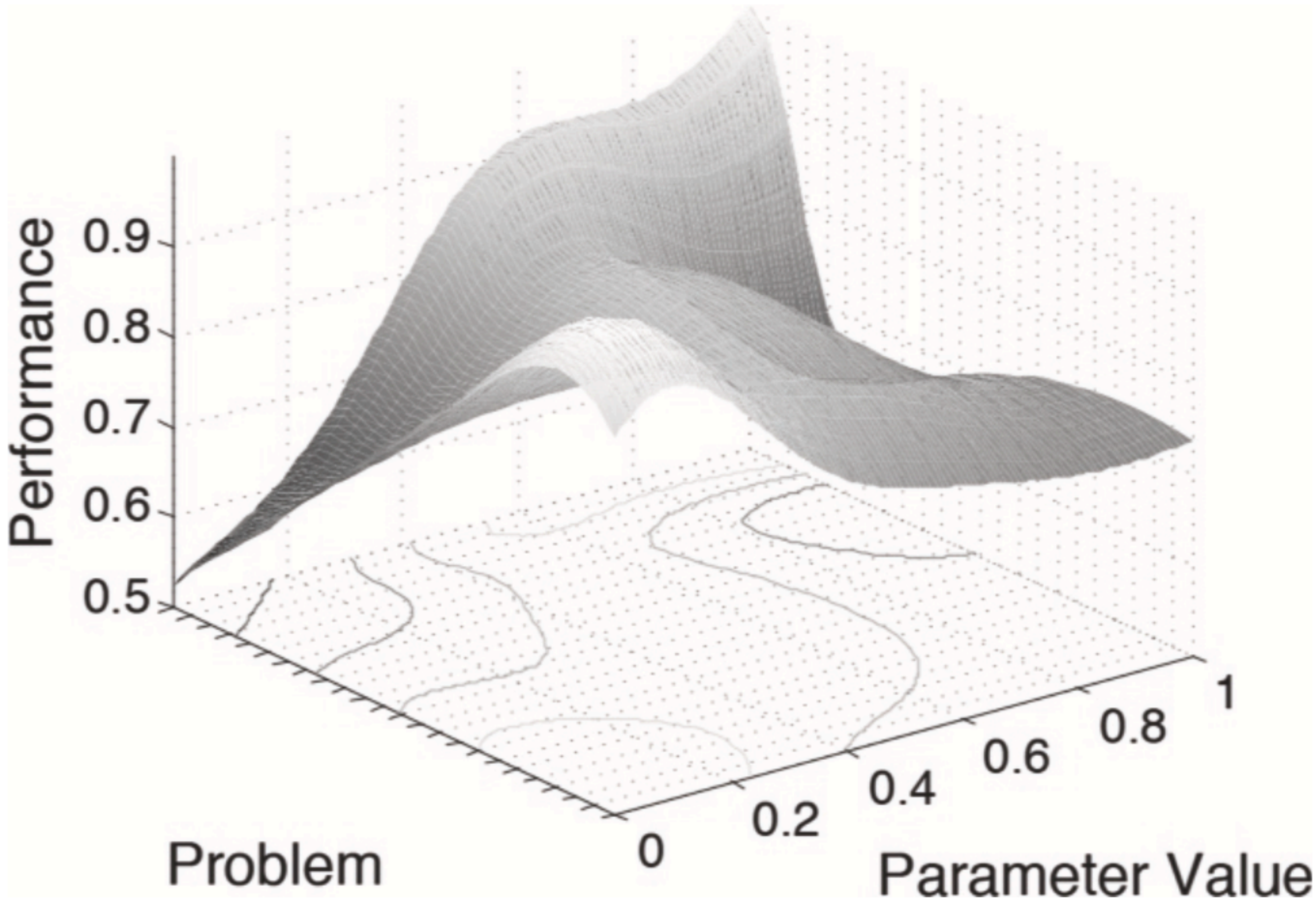
- there is a common metric for each of the three ways that performance can be measured:

metric		measurement technique
MBF	mean best fitness	fix time and measure quality
AES	average number of evaluations to a solution	fix quality and measure time
SR	success rate	fix both and measure completion

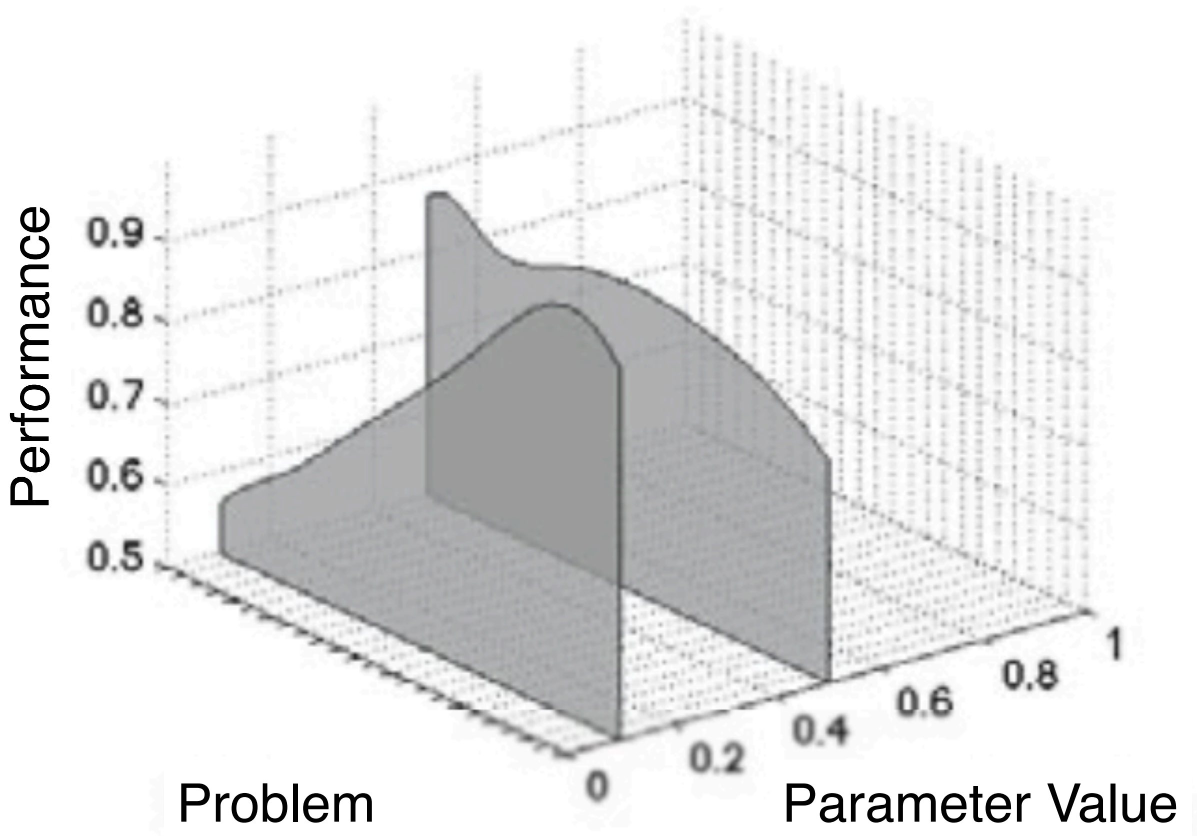
Robustness

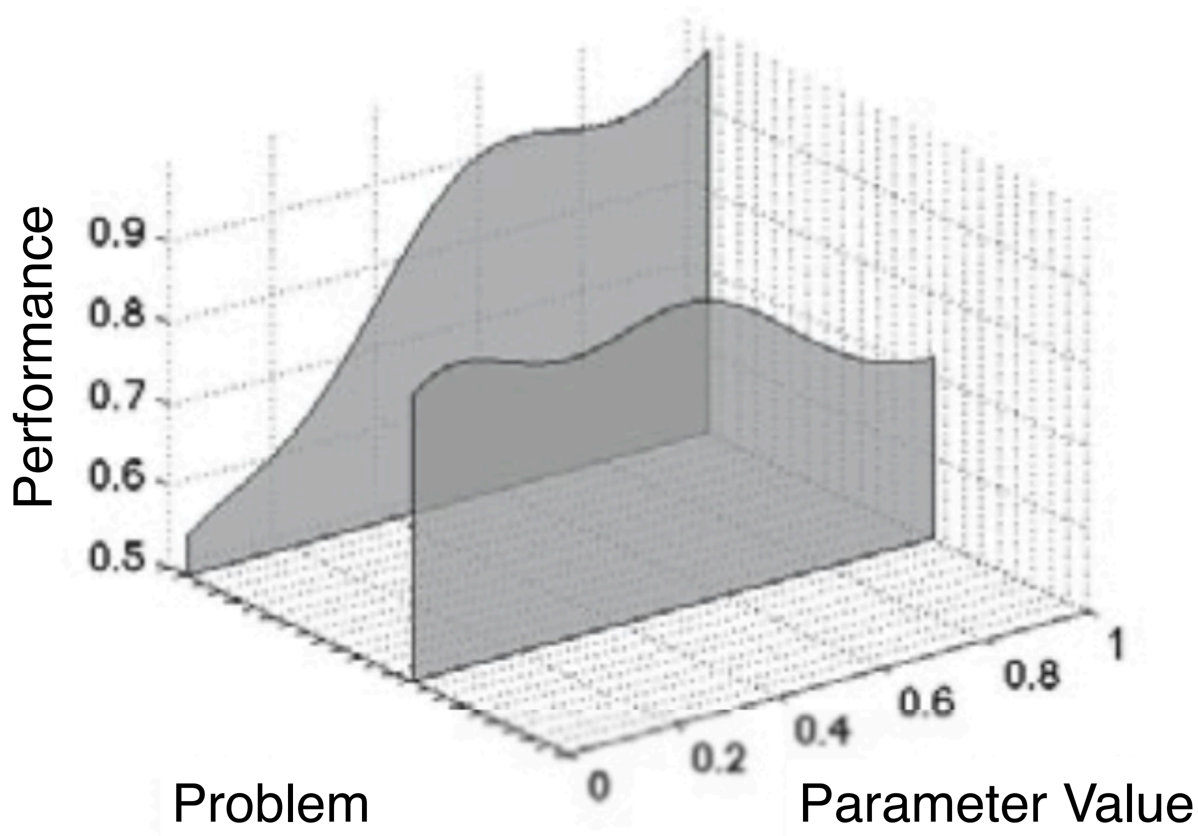
- robustness is a measure of an algorithm's performance across some dimension
- such as how well it performs depending upon the:
  - problem instance
  - parameter vector being used
  - effects of the random number generator
- context determines which type of robustness is important in a given situation
- for example if we are tuning an EA on a test suite consisting of many problem instances or test functions then we hope to see good performance across all problem instances
  - so the result of the tuning process will be a single parameter vector that provides that

Robustness: Grand Utility Landscape



Robustness: Landscape Slices





the effect of keeping a parameter's value fixed across problems

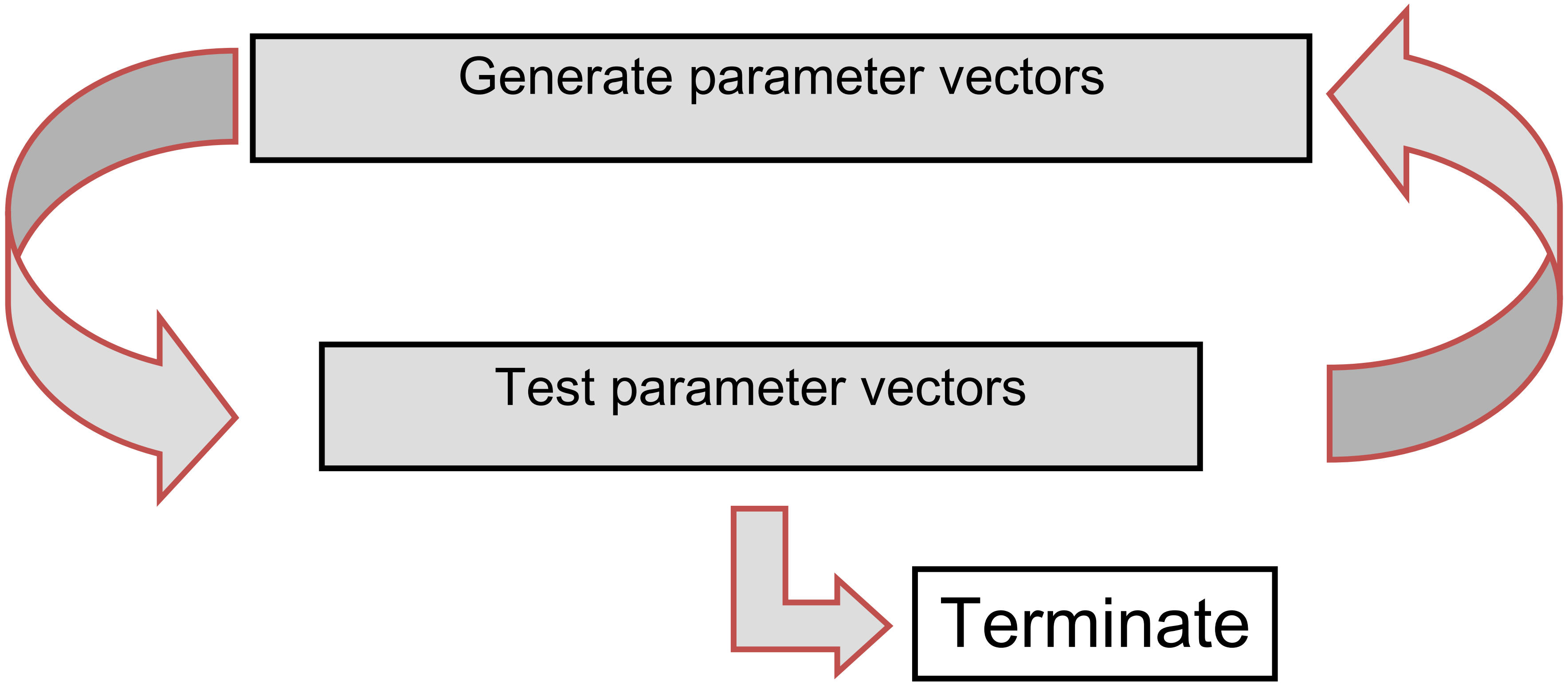
the effect of varying a parameter's value for specific problems

Offline versus Online Calibration / Design

- Design / calibration method
- offline = parameter tuning
- online = parameter control
- advantages of tuning:
  - easier
  - control strategies have parameters too
    - ⇒ would need tuning themselves
  - knowledge about tuning (utility landscapes) can help the design of good control strategies
  - there are indications that good tuning works better than control

Tuning by Generate and Test

- EA tuning is a search problem itself
- straightforward approach: GENERATE and TEST



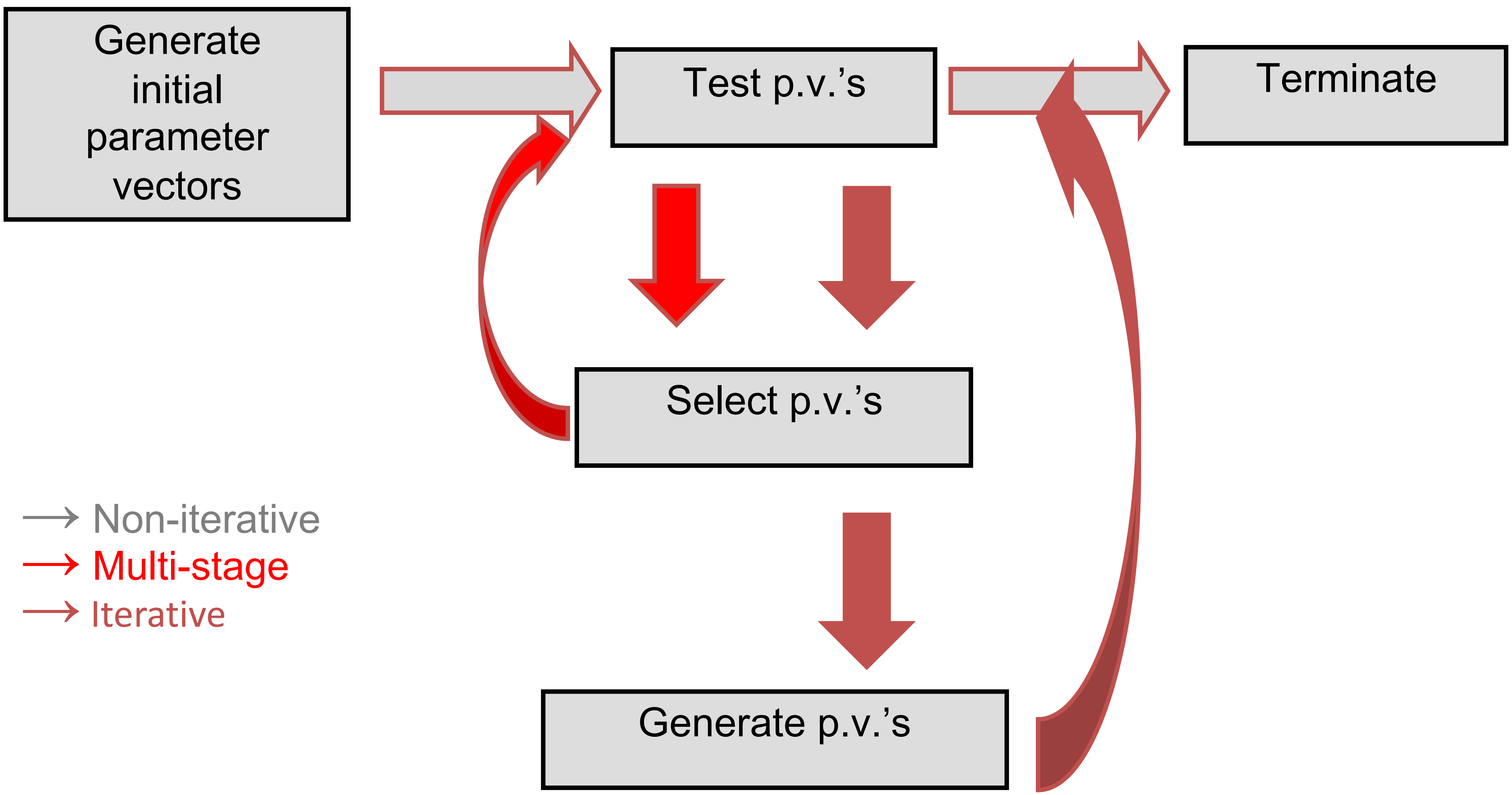
Categories of Tuners

- two main categories of tuners:
  - non iterative
  - iterative
- non-iterative:
  - execute the GENERATE step only once at initialisation
  - so use a fixed set of parameter vectors
  - each vector is tested during the test phase to find the best vector in the set
- iterative:
  - do not fix the set of vectors during initialization
  - start with a small initial set and create new vectors iteratively during execution, based on performance of existing vectors

Single and Multi-Stage Procedures

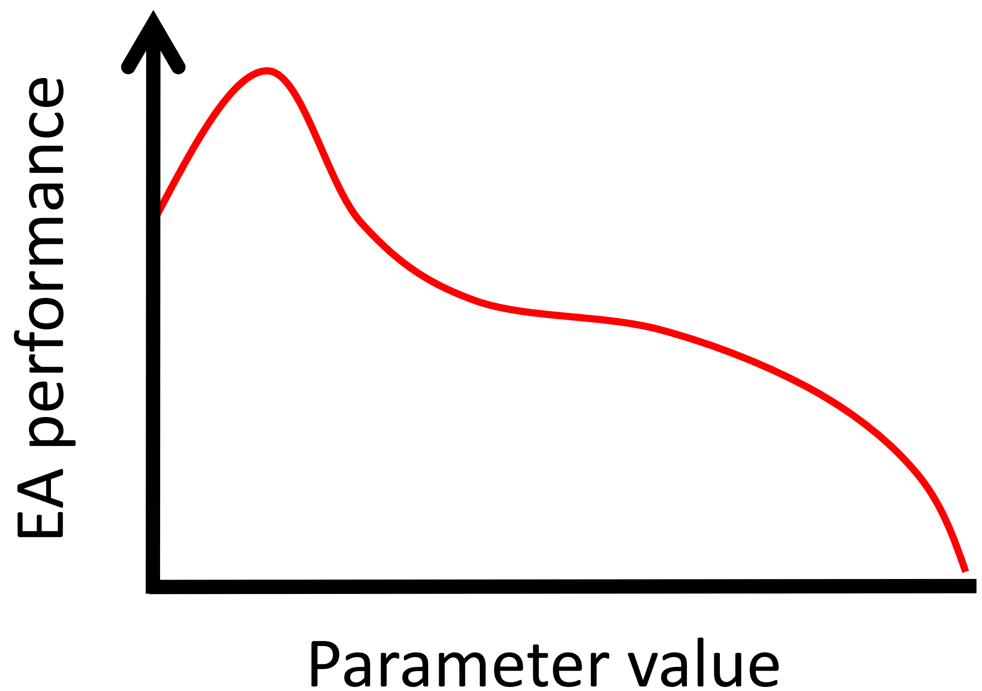
- single stage procedures perform the same number of tests for each given vector
- multi-stage procedures augment the TEST step by adding a SELECT step
- where only promising vectors are selected for further testing
- those with a low performance are ignored

Generate and Test: Under the Hood

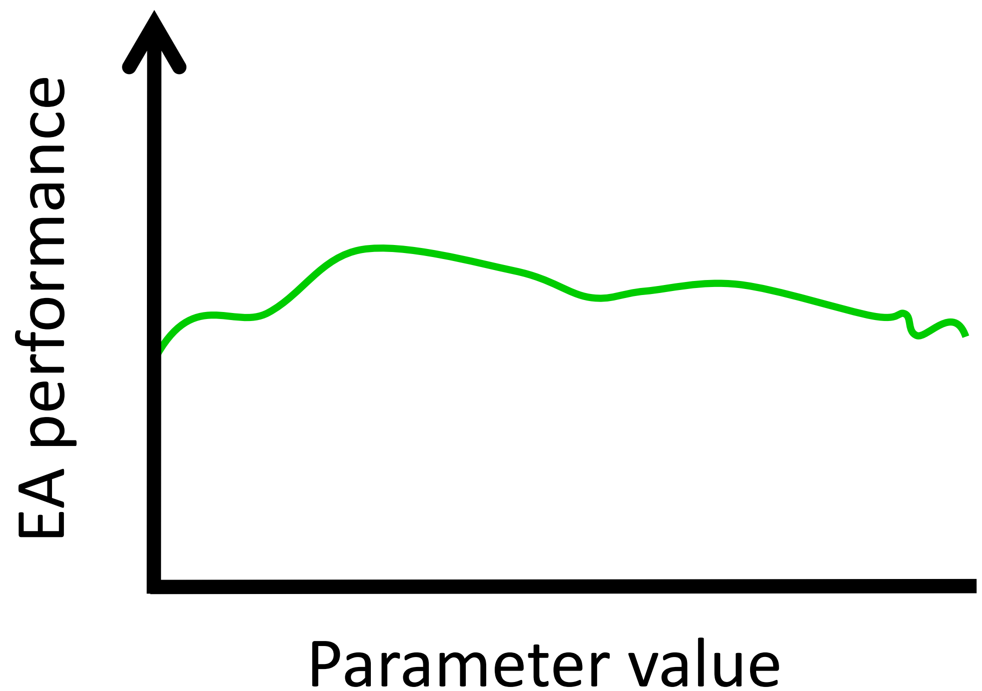


Numeric Parameters

- population size, crossover rate, tournament size, …
- domain is subset of $\mathbb{R}, \mathbb{Z}, \mathbb{N}$ (finite or infinite)
- sensible distance metric $\Rightarrow$ searchable



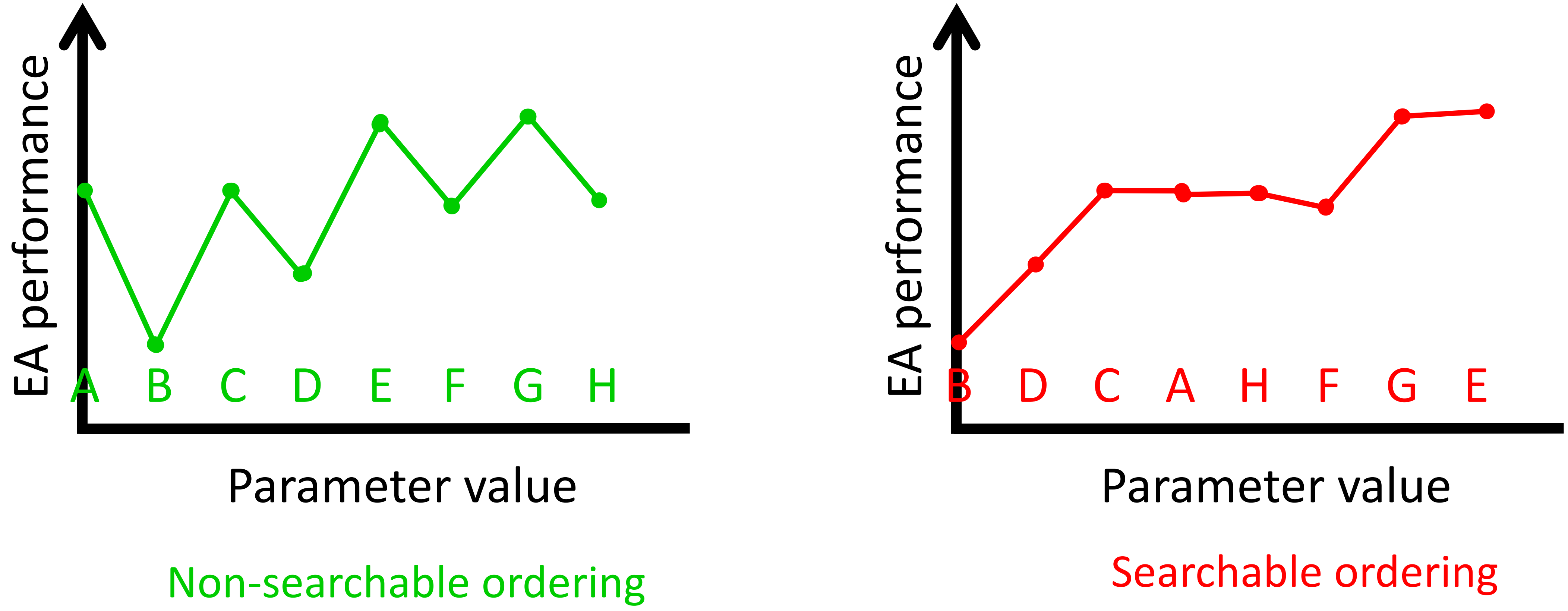
Relevant parameter



Irrelevant parameter

Symbolic Parameters

- crossover operator, elitism, selection method
- finite domain, such as `{1-point, uniform, averaging}`
- no sensible distance metric ⇒ non-searchable
    - so must be sampled



Notes on Parameters

- a value of a symbolic parameter can introduce a numeric parameter, such as:
  - selection = tournament ⇒ tournament size
  - populations type = overlapping ⇒ generation gap
- parameters can have a hierarchical, nested structure
- number of EA parameters is not defined in general
- cannot simply denote the design space / tuning search space by

$$S = Q_1 \times \ldots Q_m \times R_1 \times \ldots \times R_n$$

with Q_i / R_j as domains of the symbolic/numeric parameters

EA and EA Instances

- the distinction between symbolic and numeric parameters leads to a distinction between EAs and EA instances
- we can consider:
    - symbolic parameters as high-level, defining the essence of an evolutionary algorithm
    - numeric parameters as low-level, defining a specific variant of this EA
- so we consider two EAs to be different if they differ in one or more of their symbolic parameters
    - for example, if they use different mutation operators
- if the values are specified for all parameters, including the numeric ones then we have an EA instance
- if two EA instances differ only in some values of their numeric parameters
    - such as mutation rate and the tournament size
- then we consider them as two variants of the same EA

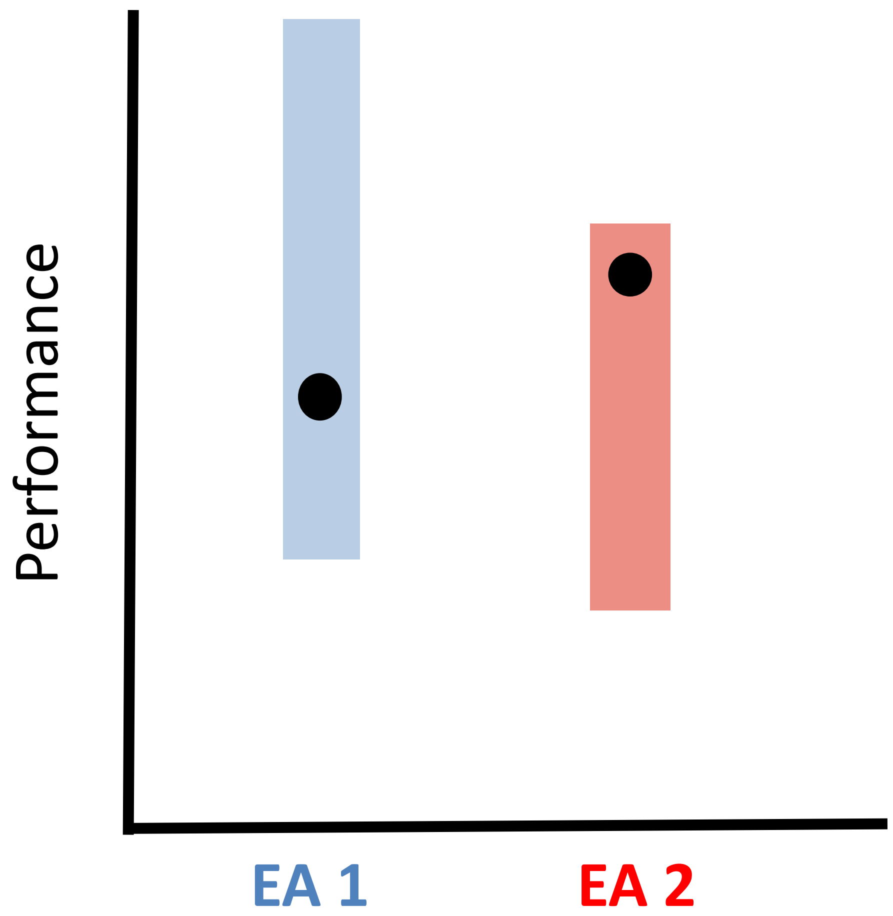
EA and EA Instances

	ALG-1	ALG-2	ALG-3	ALG-4
SYMBOLIC PARAMETERS				
Representation	Bit-string	Bit-string	Real-valued	Real-valued
Overlapping pops	N	Y	Y	Y
Survivor selection	–	Tournament	Replace worst	Replace worst
Parent selection	Roulette wheel	Uniform determ	Tournament	Tournament
Mutation	Bit-flip	Bit-flip	N(0,σ)	N(0,σ)
Recombination	Uniform xover	Uniform xover	Discrete recomb	Discrete recomb
NUMERIC PARAMETERS				
Generation gap	–	0.5	0.9	0.9
Population size	100	500	100	300
Tournament size	–	2	3	30
Mutation rate	0.01	0.1	–	–
Mutation stepsize	–	–	0.01	0.05
Crossover rate	0.8	0.7	1	0.8

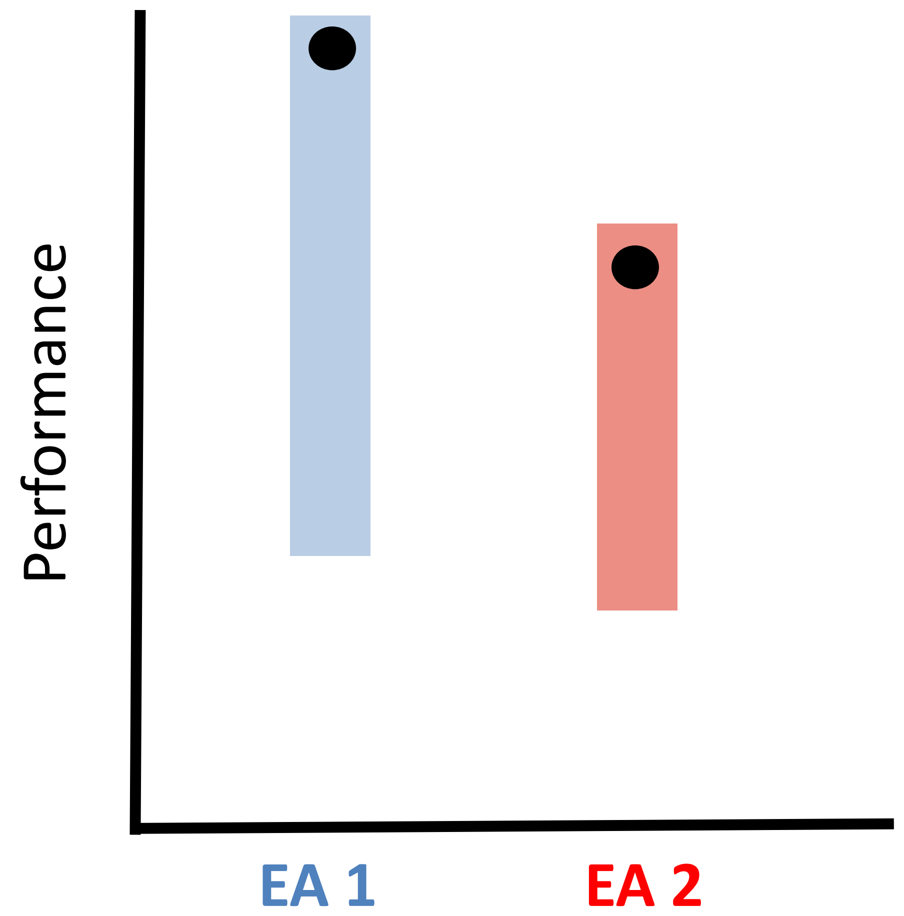
Which Tuning Method?

- differences between tuning algorithms
    - maximum utility reached
    - computational costs
    - number of their own parameters – overhead costs
    - insights offered about EA parameters
        - such as probability distribution, interactions, relevance, explicit model…
- similarities between tuning algorithms
    - nobody is using them
    - can find good parameter vectors
- solid comparison is missing – but work is ongoing

Tuning versus Not-Tuning



EA as is (accidental parameters)

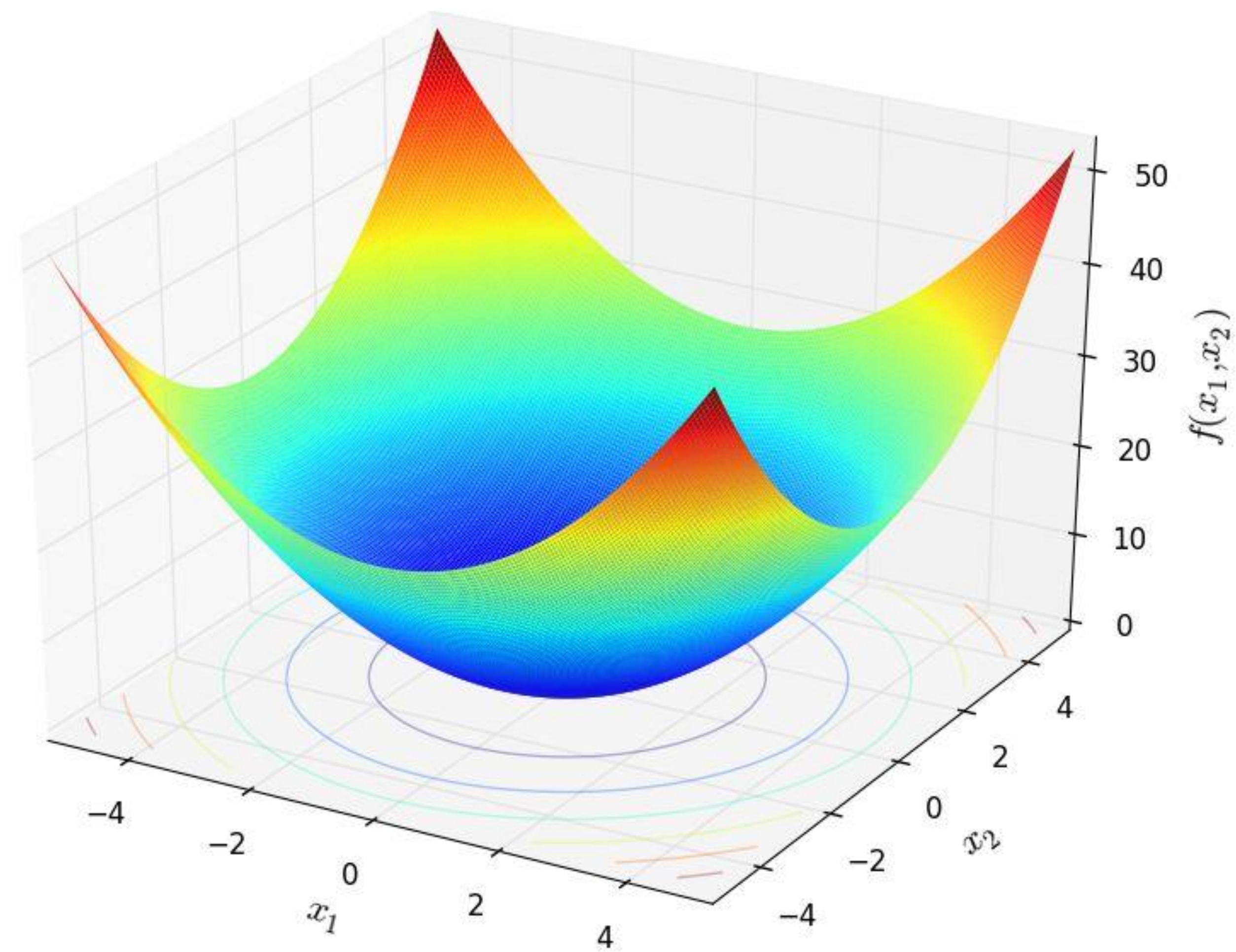


EA as it can be (“optimal” parameters)

Example Study: 'Best Parameters'

setup:

- problem: Sphere Function
- EA: defined by Tournament Parent Selection, Random Uniform Survivor Selection, Uniform Crossover, BitFlip Mutation
- tuner: REVAC:
  - “Relevance Estimation and Value Calibration”
  - a heuristic generate-and-test method
  - iteratively adapts a set of parameter vectors of a given EA



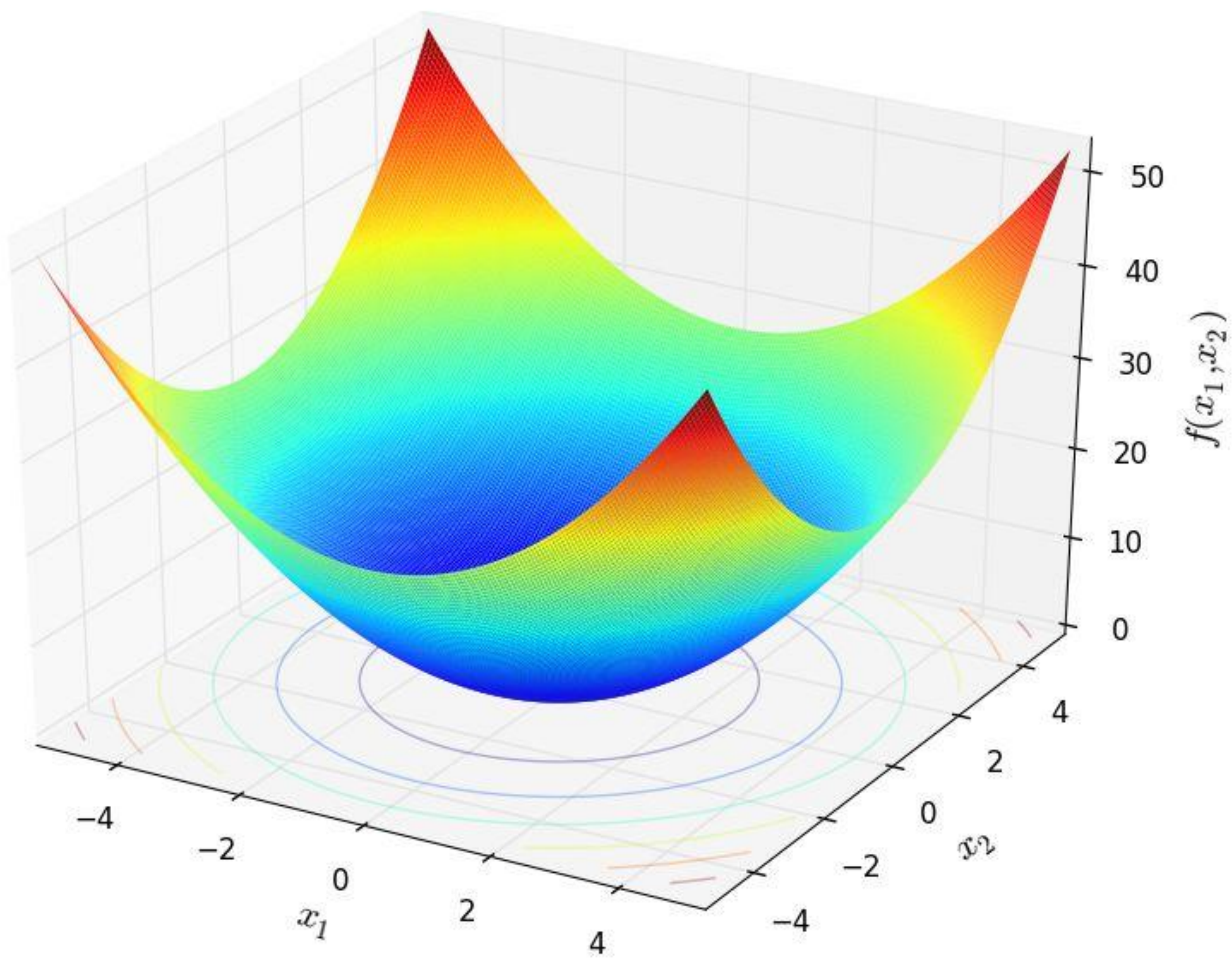
Example Study: 'Best Parameters'

results:

- best EA had the following parameter values
  - population Size: 6
  - tournament Size: 4
  - ...

conclusion:

- *for this problem* we need a high (parent) selection pressure
- probably because the problem is unimodal



Example Study: 'Good Parameters'

setup:

- same as before

results:

  - the 25 best parameters vectors have their values within the following ranges
  - mutation rate: `[0.01, 0.011]`
  - crossover Rate: `[0.2, 1.0]`
  - …

conclusion:

- *for this problem* the mutation rate is much more relevant than the crossover rate

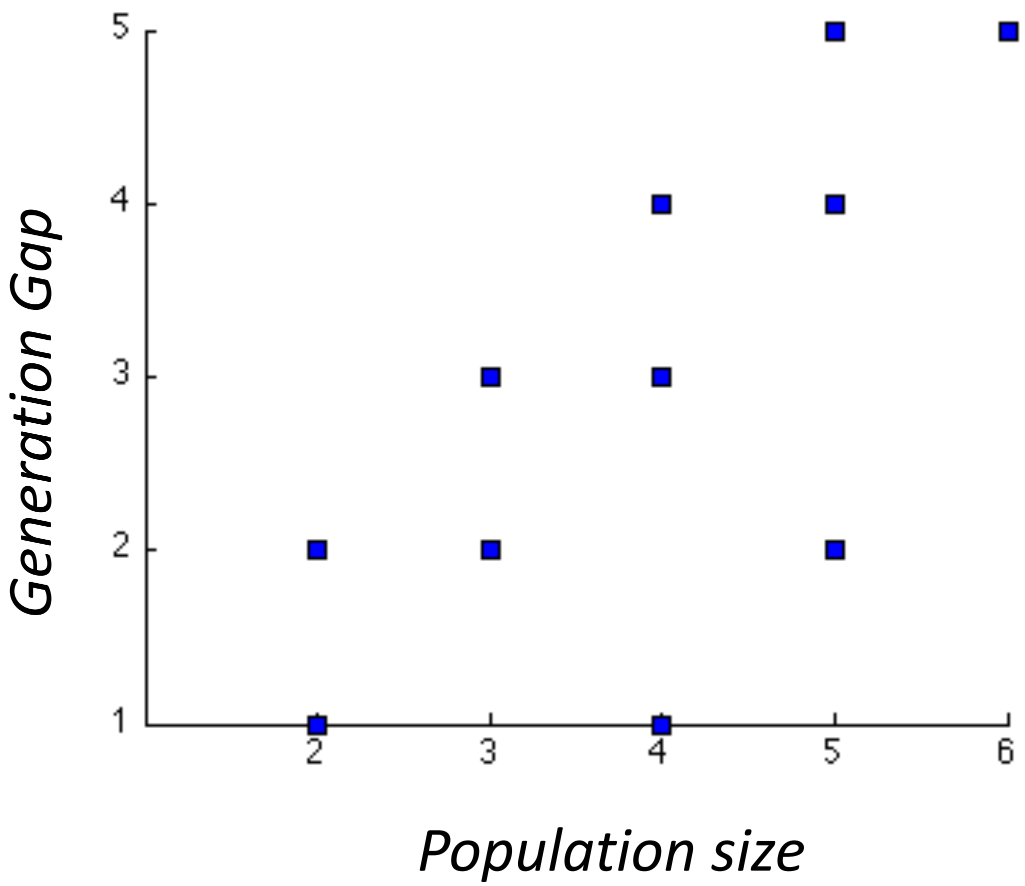
Example Study: 'Interactions'

setup:

- same as before

results:

  - plotting the population size and generation gap of the best parameter vectors shows the following



conclusion:

- *for this problem* the best results are obtained when (almost) the complete population is replaced every generation

Recommendations

- do tune your evolutionary algorithm
- be aware of any magic constants
- decide: speed or solution quality?
- decide: specialist of generalist EA?
- measure and report tuning effort
- try out Eiben & Smith’s toolbox:

  http://sourceforge.net/projects/mobat

Time for a Change of Culture?

- fast and good tuning can lead to new attitude
- past & present: robust EAs preferred
- future: problem-specific EAs preferred
- old question: what is better the GA or the ES?
- new question: what symbolic configuration is best?

  … given a maximum effort for tuning
- new attitude / practice:
  - tuning efforts are measured and reported
  - EAs with their practical best settings are compared, instead of unmotivated ‘magical’ settings

Reading & References

- slides based on and adapted from, Chapter 7 (and slides) of Eiben & Smith's *Introduction to Evolutionary Computing*
- see Brightspace resources for Eiben, Hinterding and Michaelwicz's in-depth article on Parameter Tuning and Control