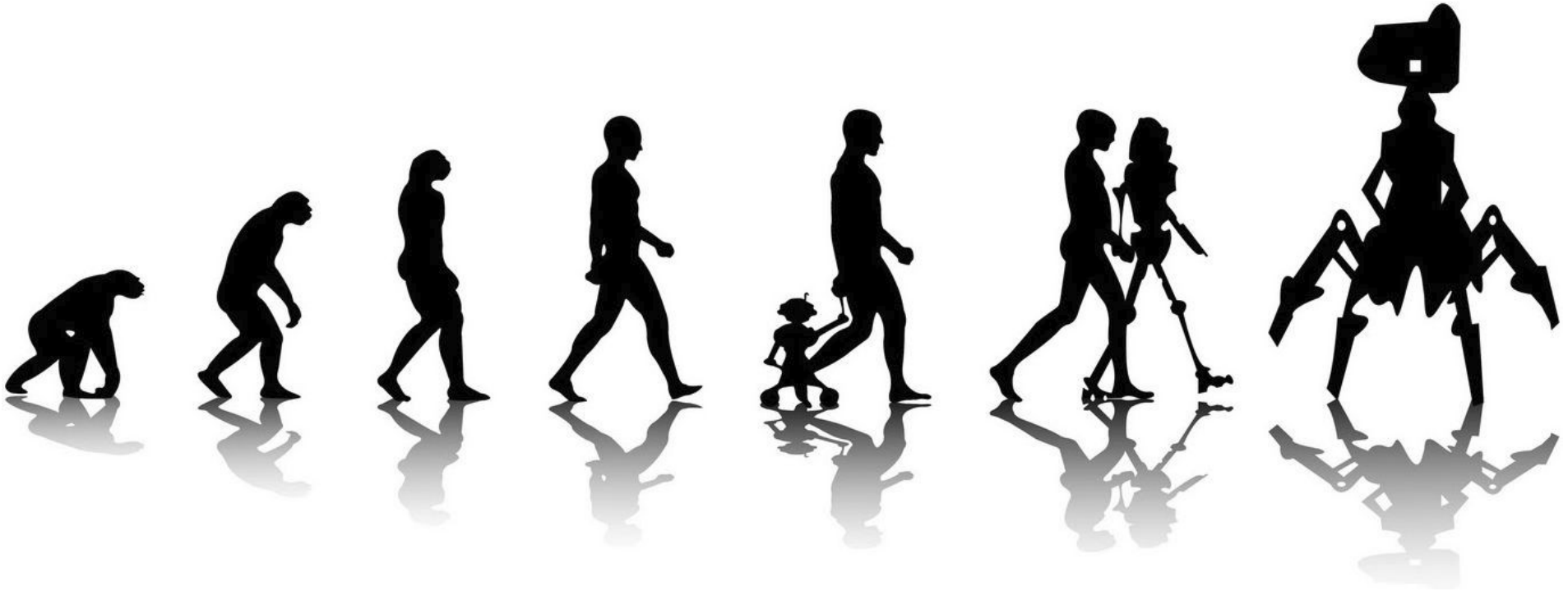


Modern Robotics: Evolutionary Robotics

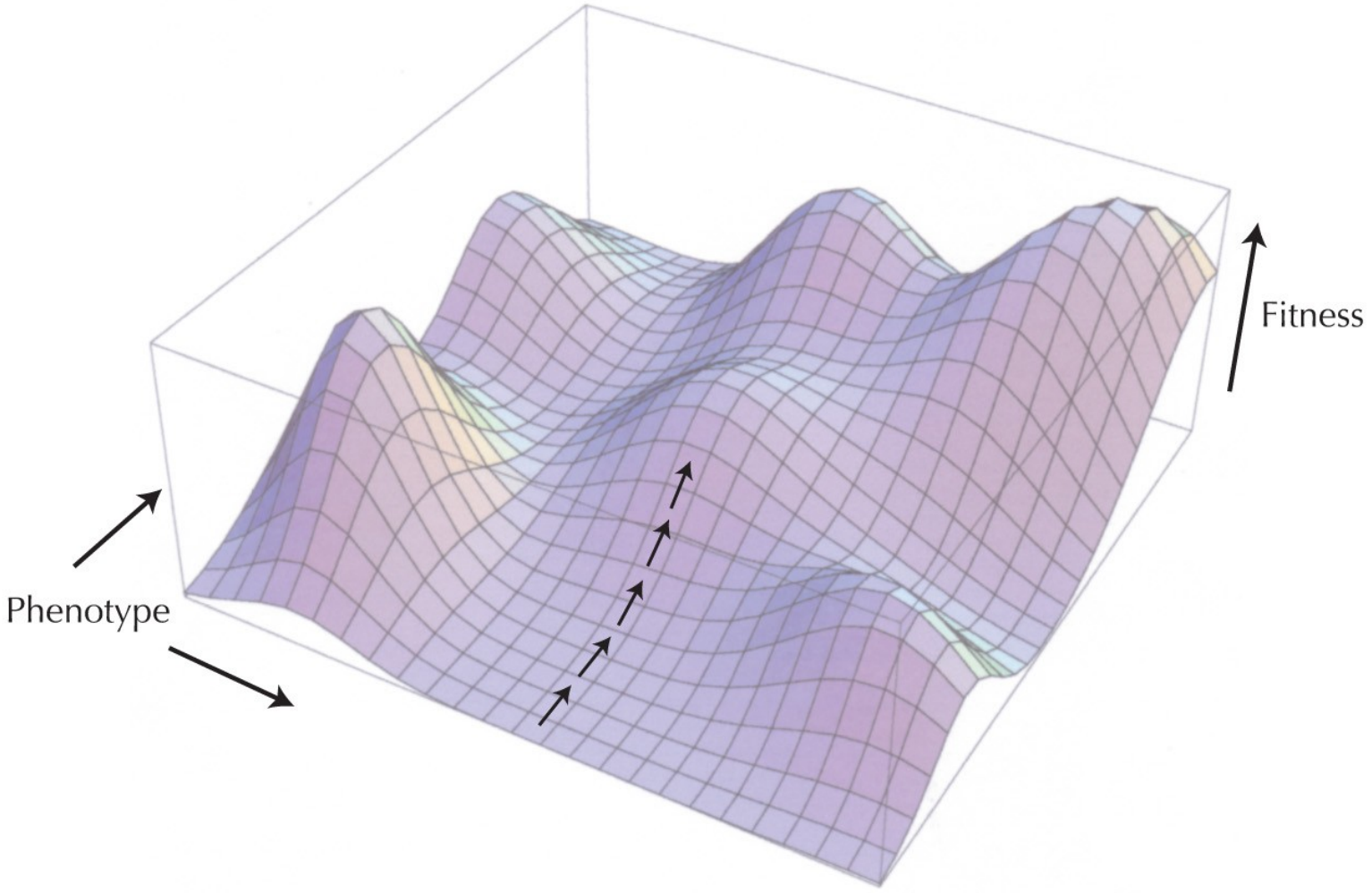
COSC 4560 / COSC 5560

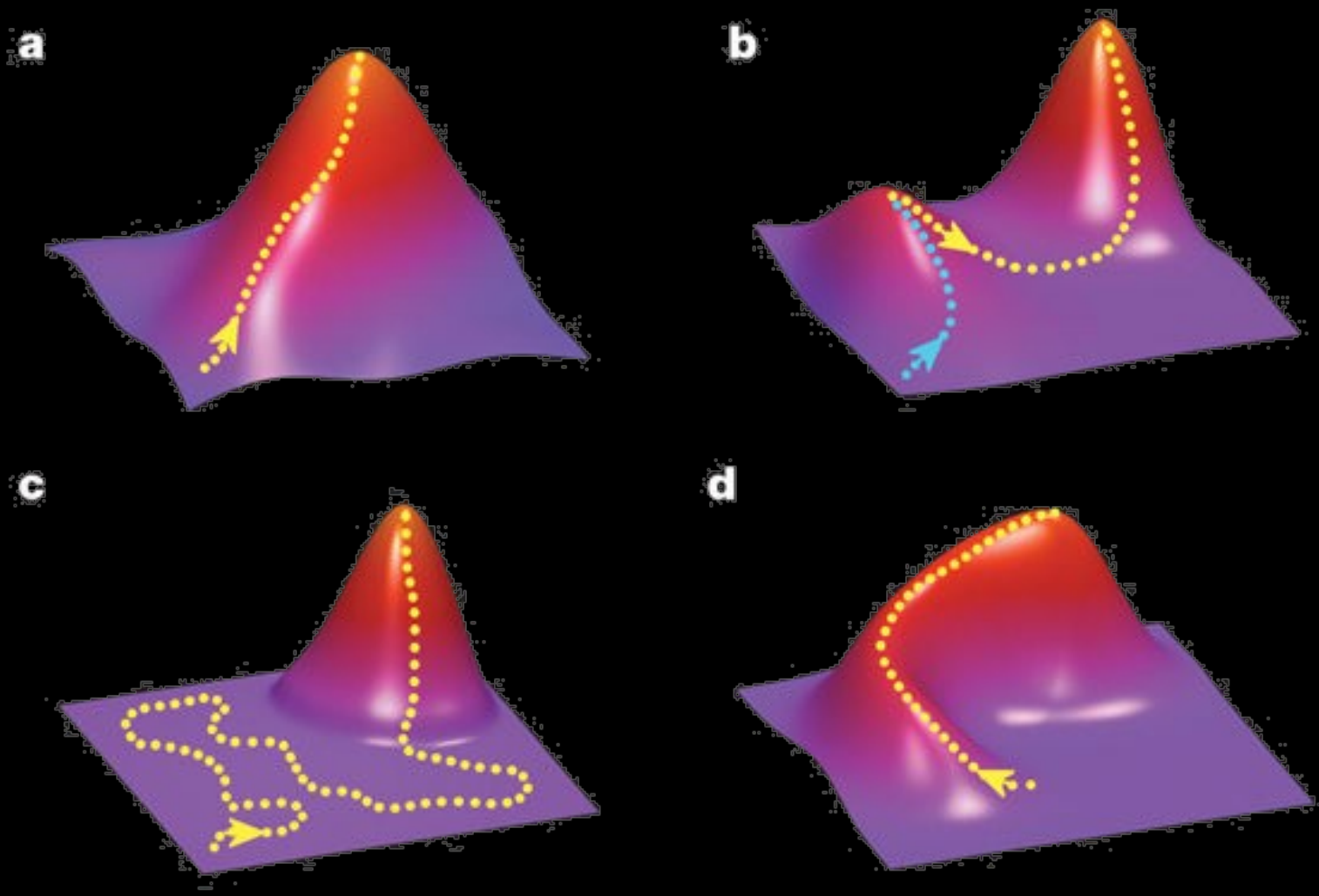
Professor Cheney

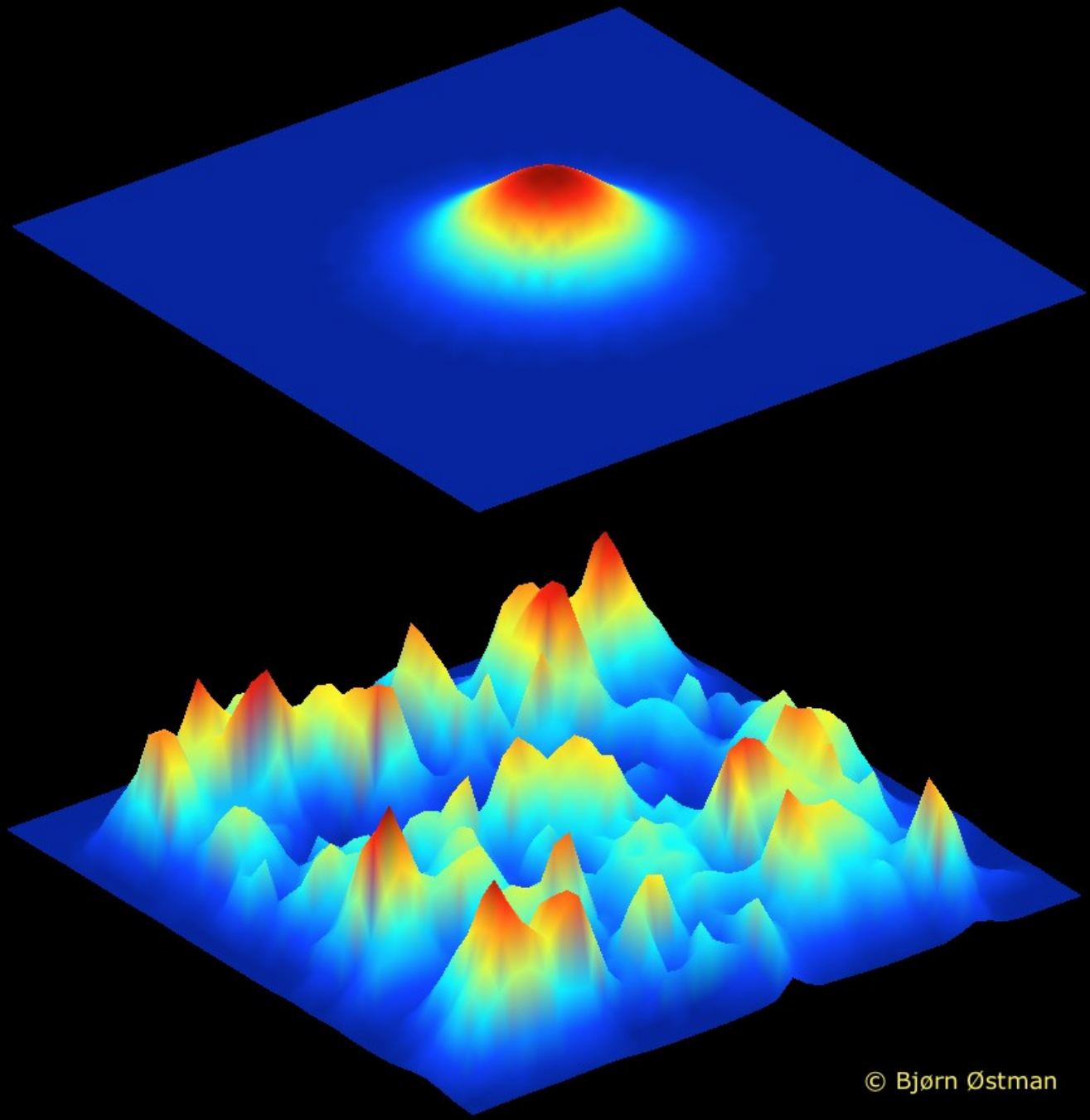
1/29/18

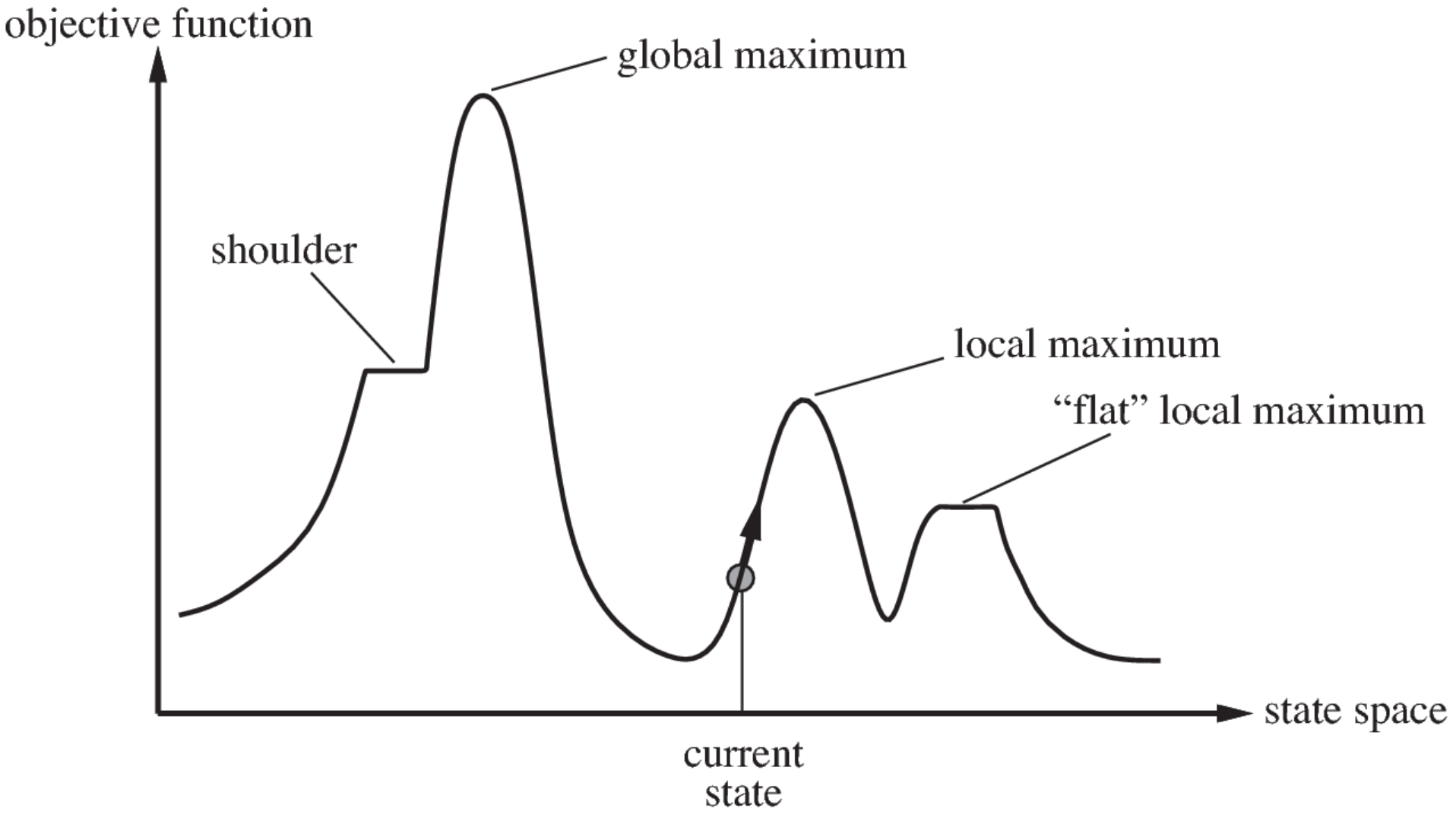
Variations of Evolutionary Algorithms

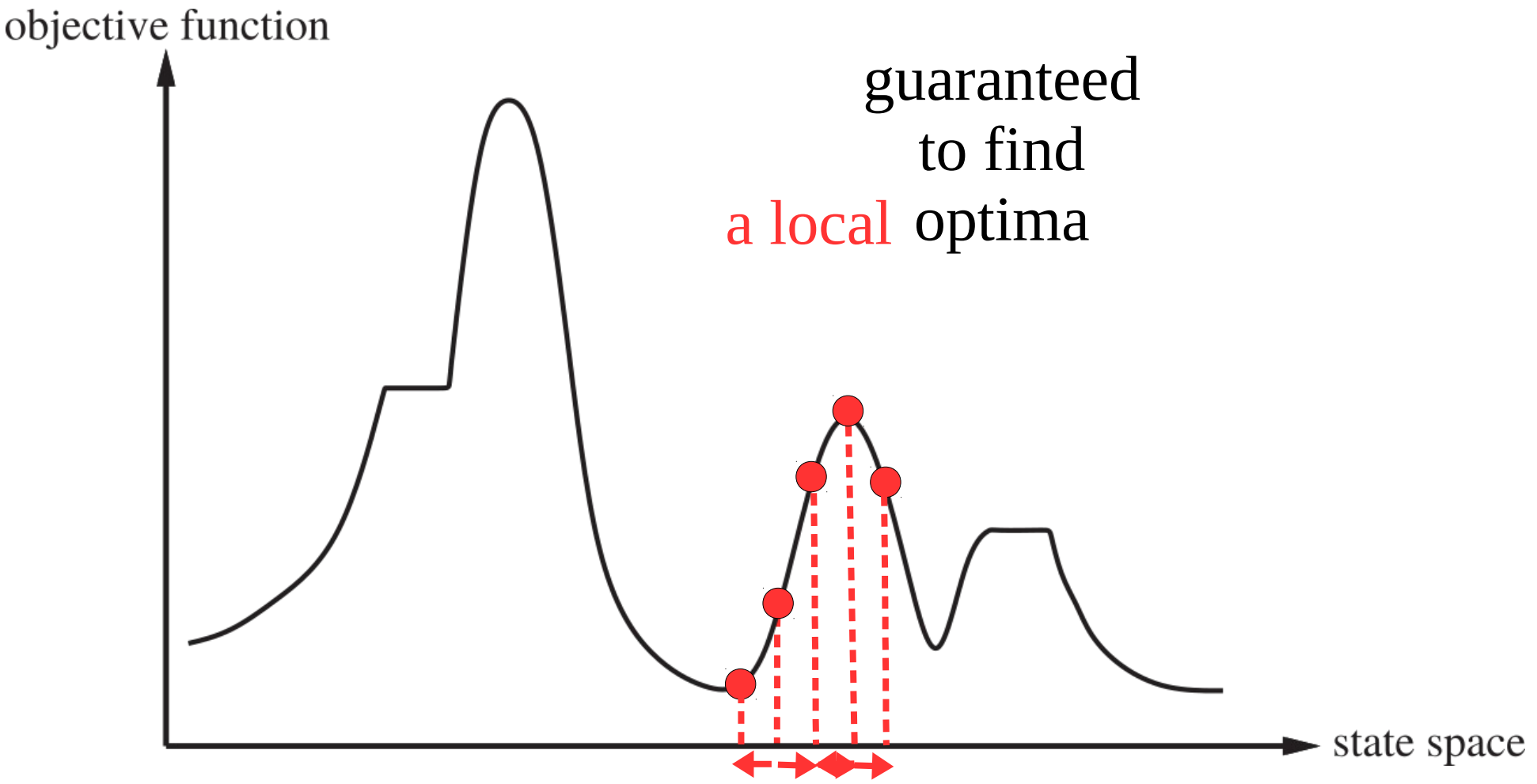
Fitness/Search Landscapes


Fitness
Phenotype


a
b
c
d

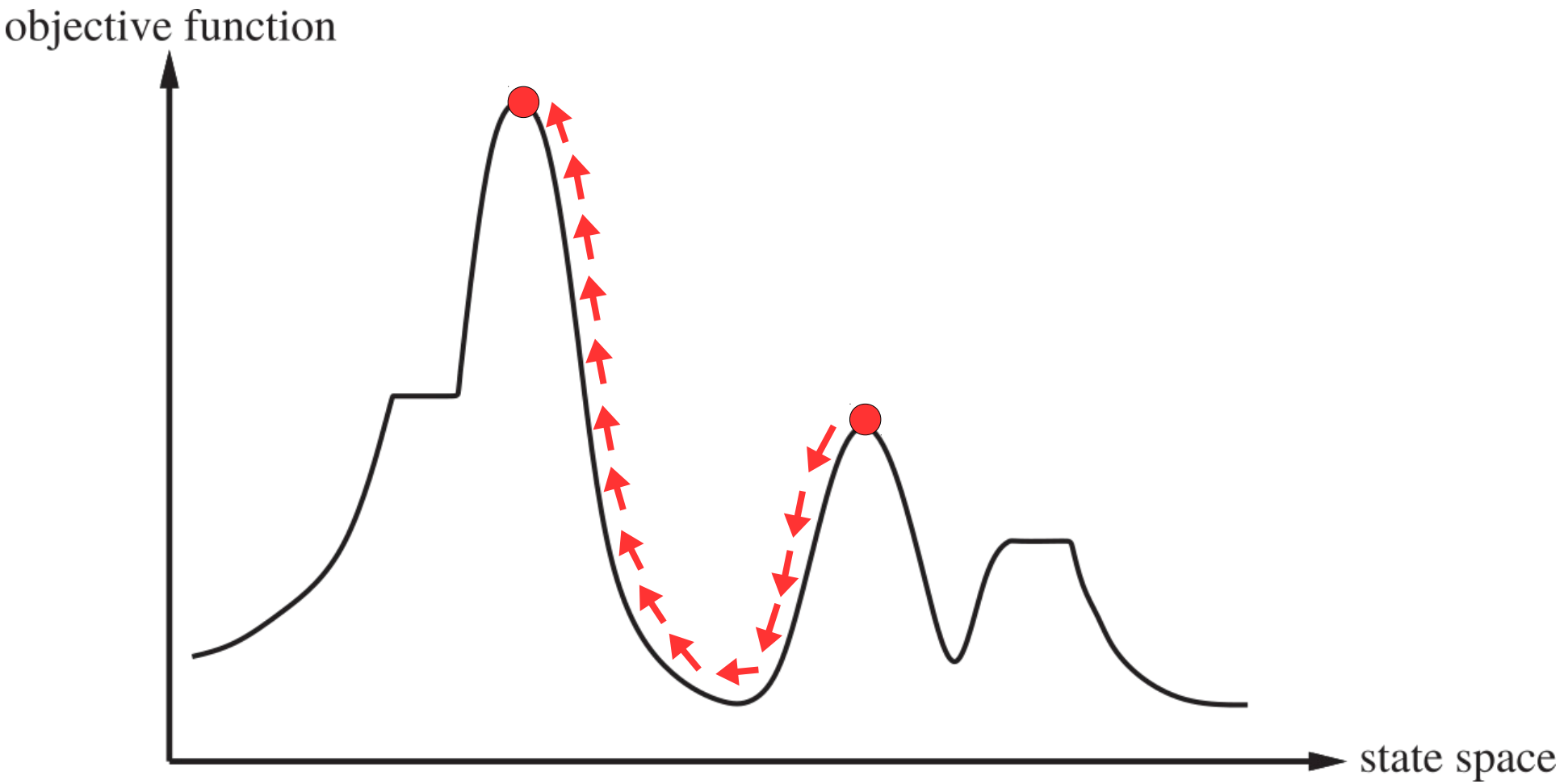

© Bjørn Østman


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state


objective function
guaranteed
to find
a local optima
state space

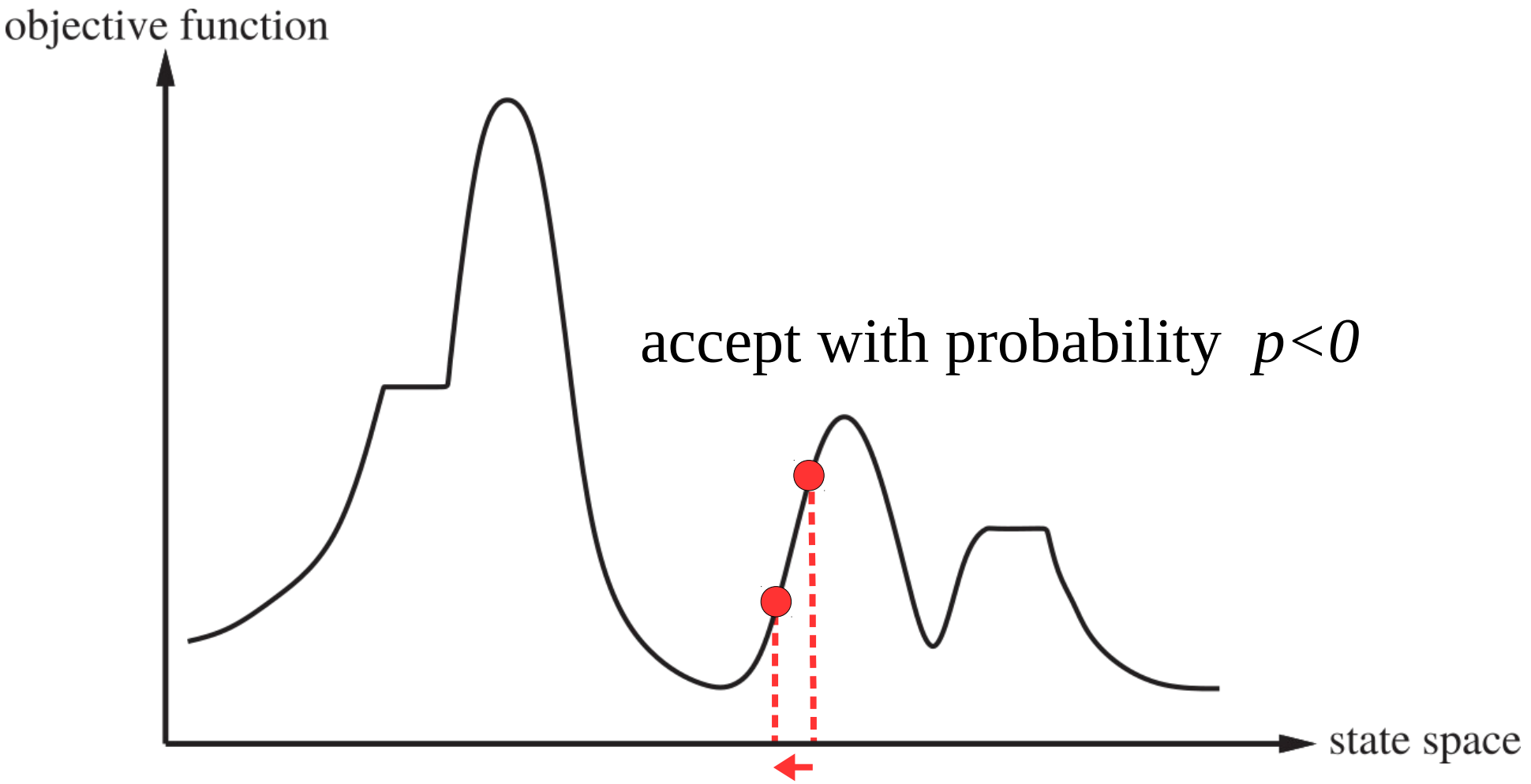
but wouldn't we rather have a global optima?

to do so, we'd have to accept (many) negative mutations to get to the better “fitness peak”


objective function
state space

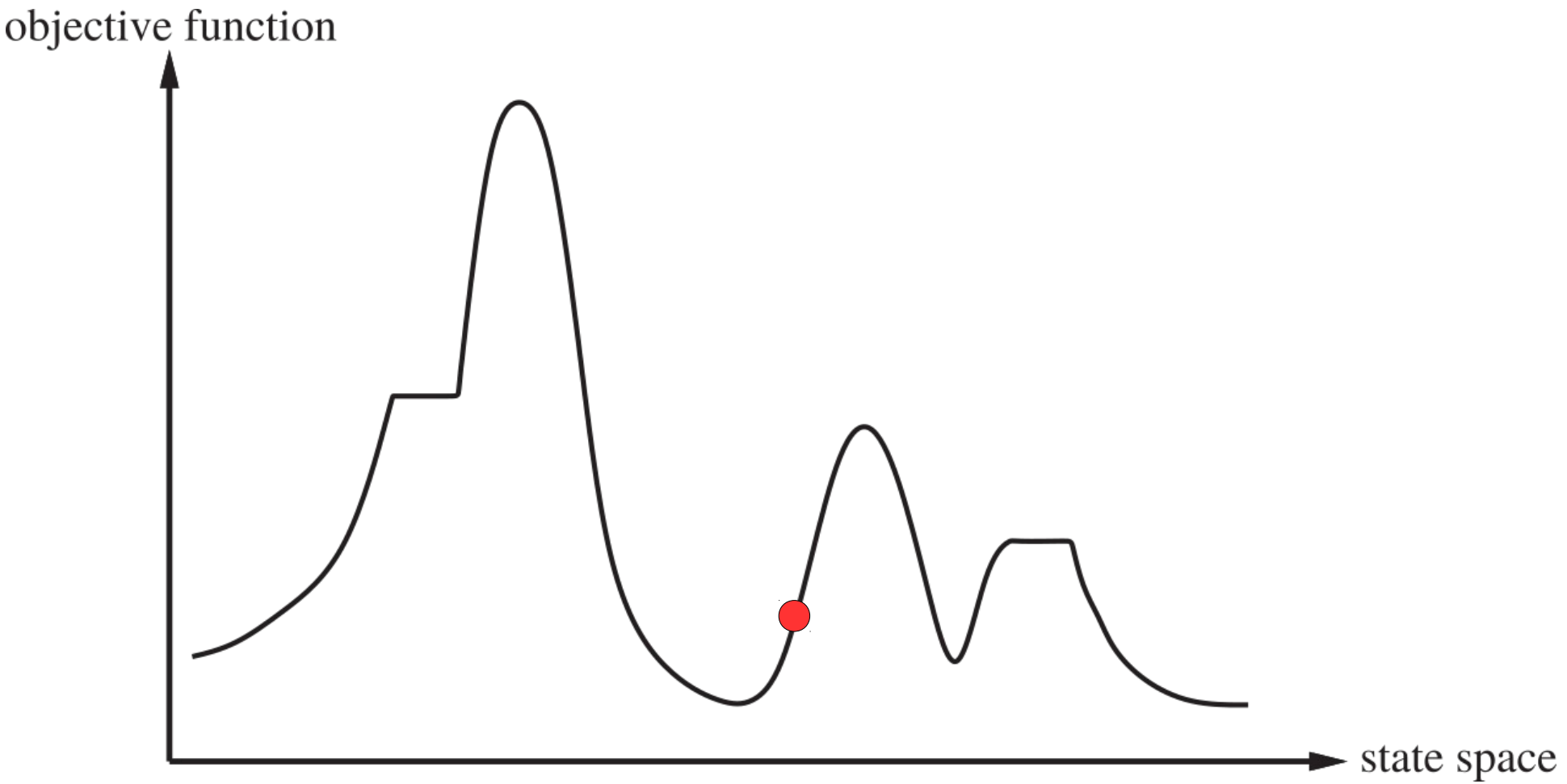
what if just sometimes,
we accepted negative mutations?

stochastic hillclimbing

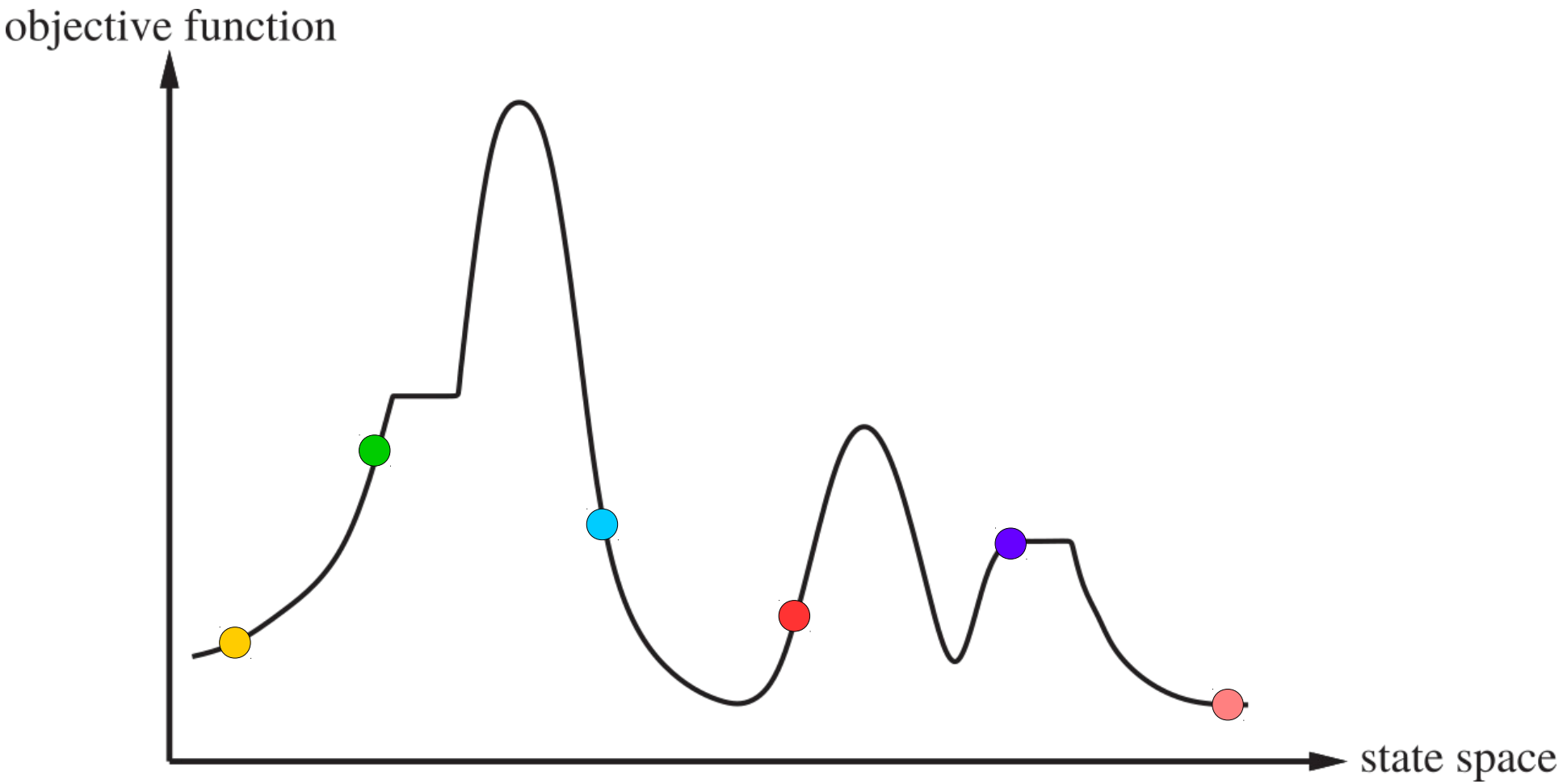


what if instead we used
random initialization?

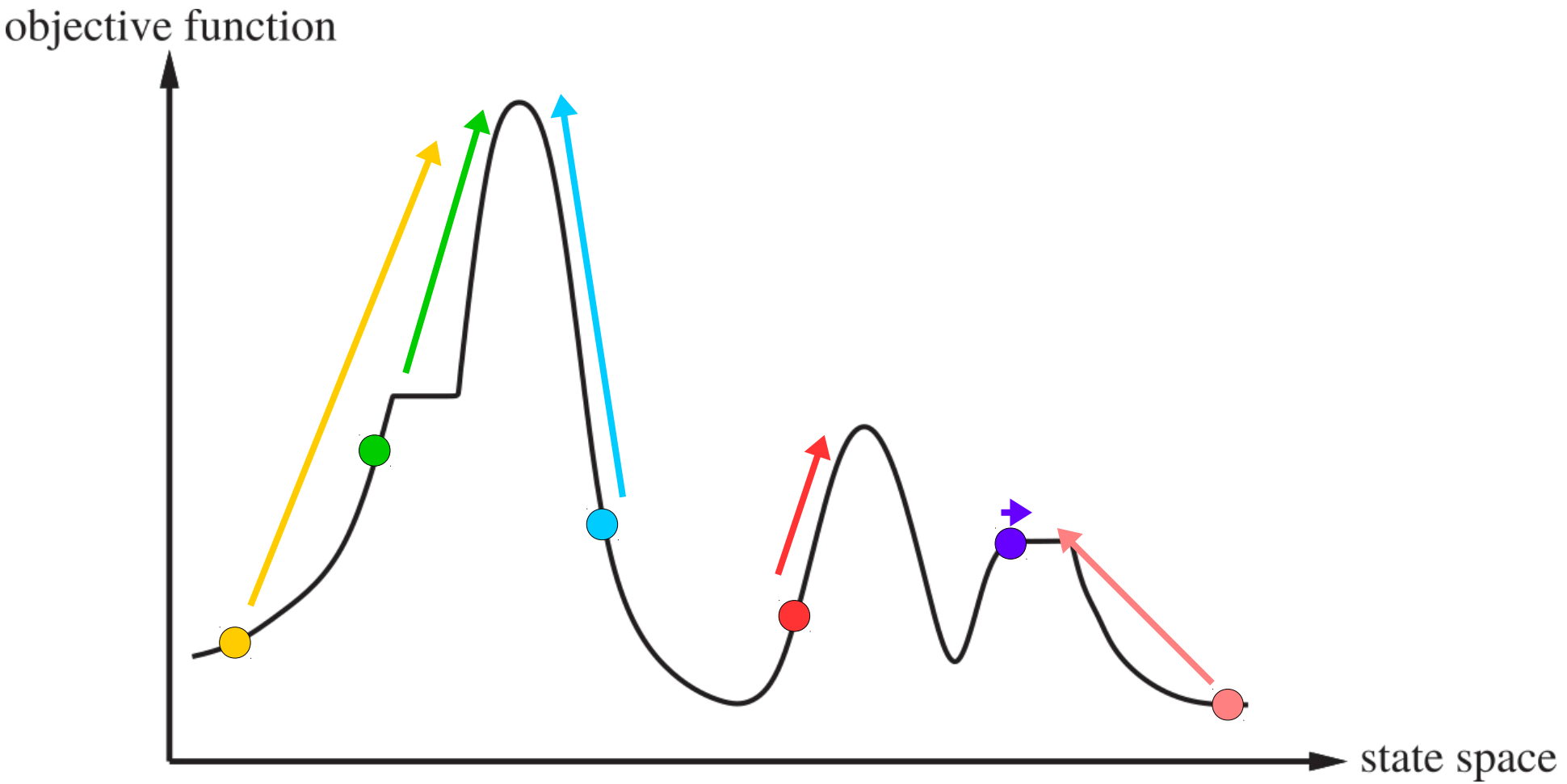
random-restart hillclimbing



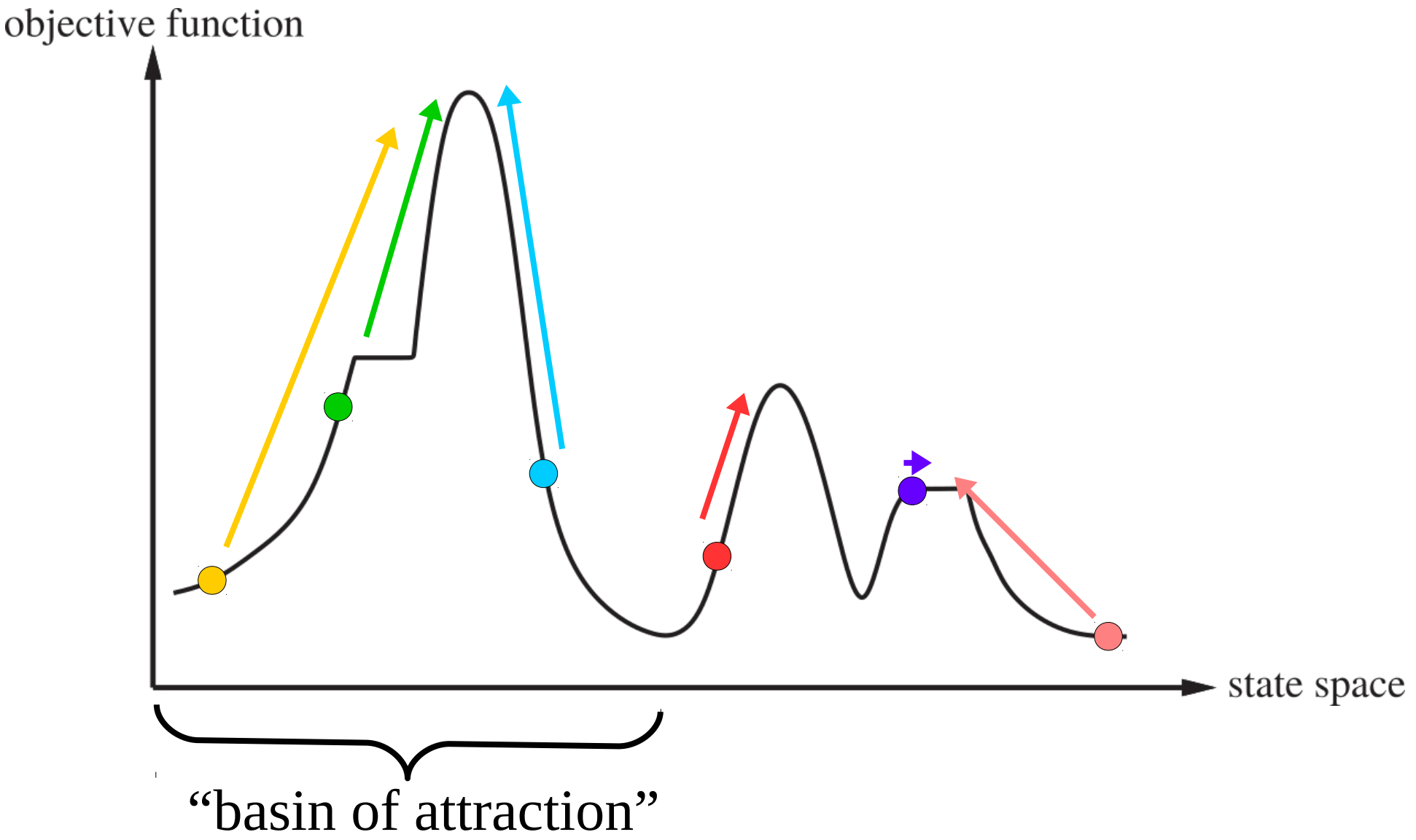
random-restart hillclimbing



random-restart hillclimbing



random-restart hillclimbing



how do you decide the probability of
(when to) accept a negative mutation or not?

(i.e. when should you explore
and when should you exploit?)

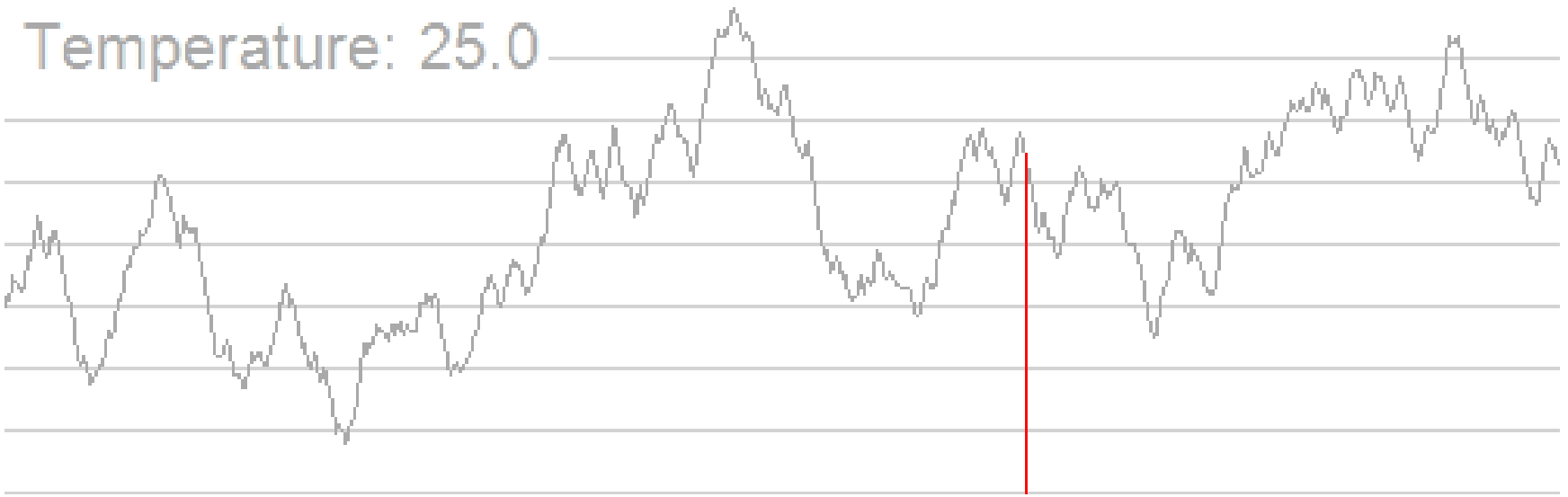
explore (take risky moves) early in search
so that you have more time to
catch up (and exploit) later on

simulated annealing

the probability of accepting a new negative mutation decreases over optimization time

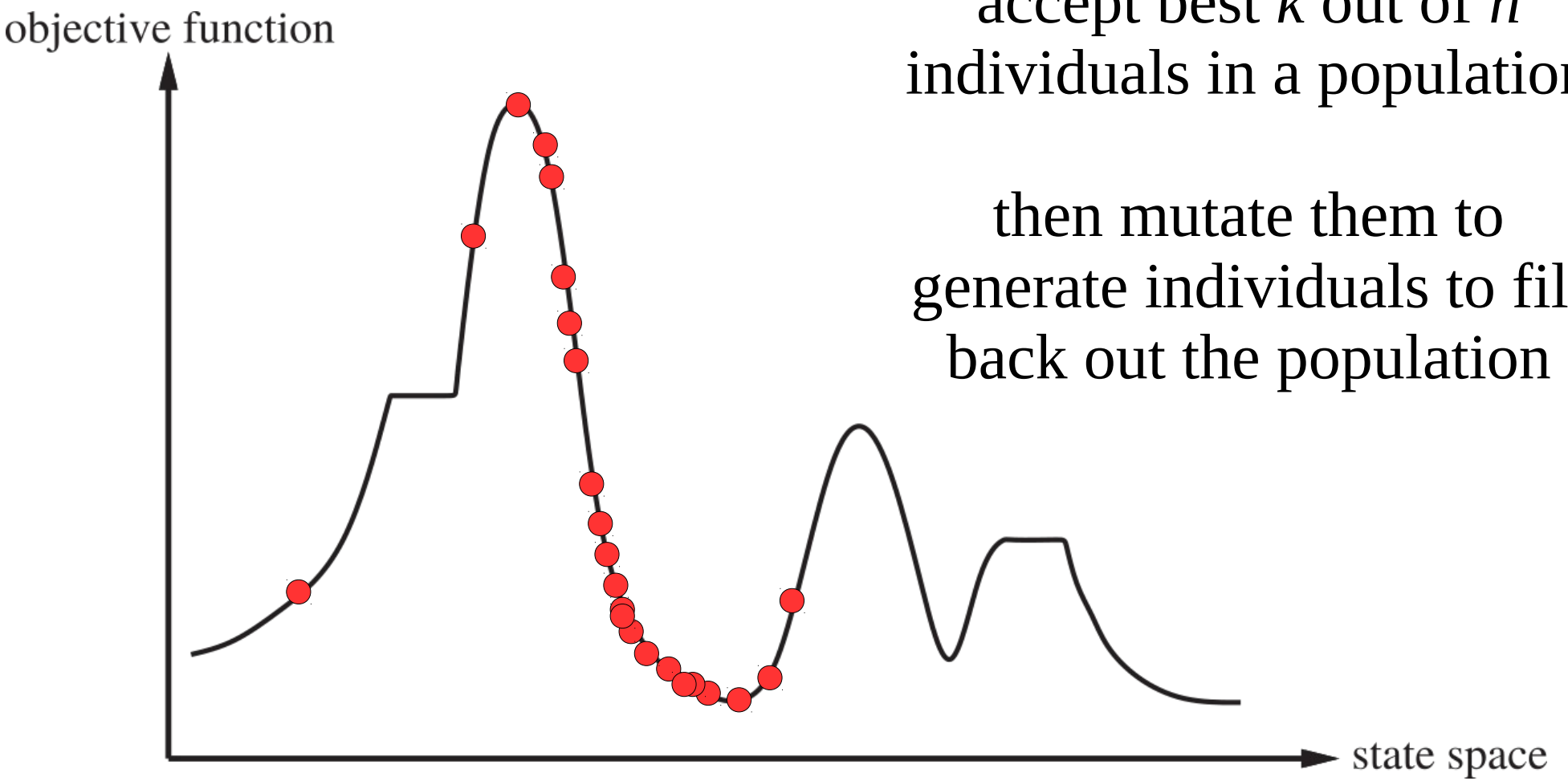
at first the point (current state) is randomly moving around all over the place (state space)

but as the system “temperature cools” over time, it settles down, and resists change
and will only accept a new position if it's an improvement over the current one


Temperature: 25.0

genetic algorithms

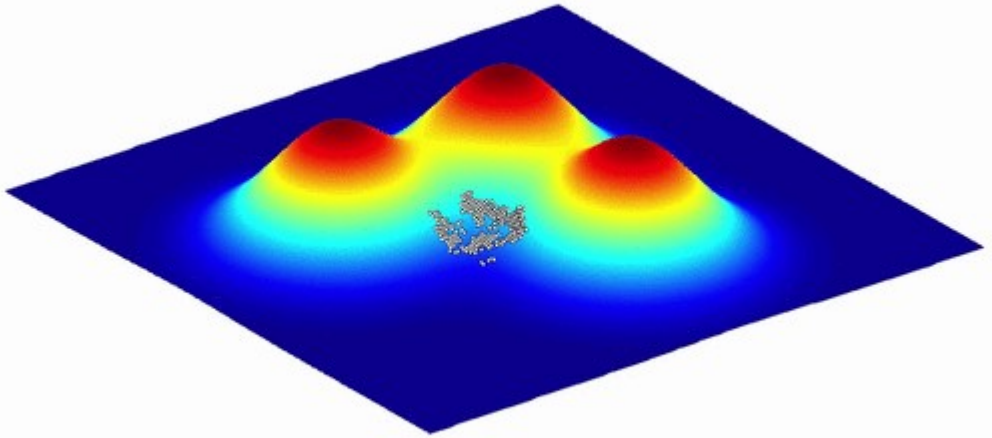
population based methods



accept best *k* out of *n* individuals in a population

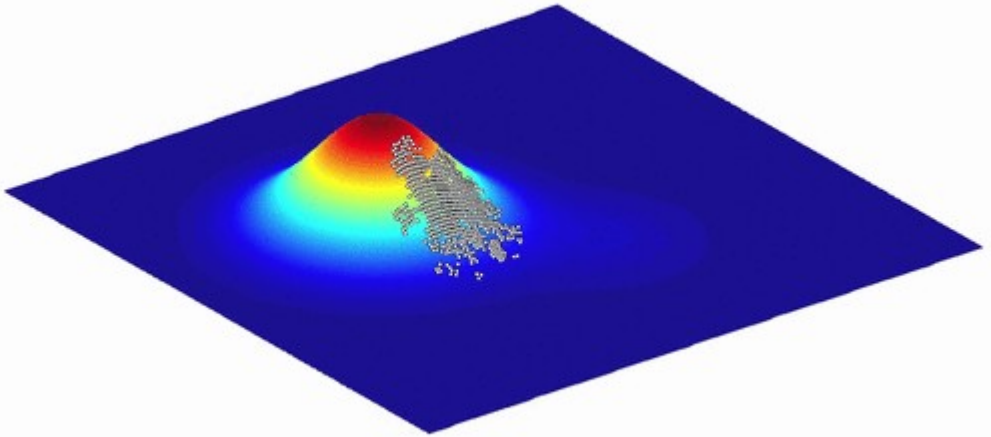
then mutate them to generate individuals to fill back out the population

Static fitness landscape



Population size, N = 2,304
Mutation rate, μ = 0.05 per trait

Dynamic fitness landscape



Population size, N = 2,304
Mutation rate, μ = 0.5 per trait

Selection Pressure

How much “pressure” is there on a solution to make an uphill move (“exploitation”) vs. a downhill move (“exploration”)?

As we saw from random-restart hillclimbing and simulated annealing, we want to explore earlier in optimization (to find the basin of our global optima), and exploit later in optimization (to climb to the optima of that basin)

Easier said than done…

Truncation Selection

Select the top-N individuals

Fit([1 0 1 0 1 1 1 0 1 1]) = 7
Fit([1 1 1 0 1 0 0 1 0 1]) = 6
Fit([0 1 1 0 1 1 0 0 1 0]) = 5

~~Fit([0 0 1 1 0 1 0 1 0 0]) = 4~~
~~Fit([0 0 0 0 1 0 1 1 1 0]) = 4~~
~~Fit([1 1 1 0 0 1 0 1 1 0]) = 4~~

Maximum selection pressure (always moving uphill!)

Fitness-Proportional Selection

Select N individuals, with an individual being selected at each draw by the proportion of fitness it represents (often without replacement)

Fit([1 0 1 0 1 1 1 0 1 1]) = 7		7/30 = 23%
Fit([1 1 1 0 1 0 0 1 0 1]) = 6		6/30 = 20%
Fit([0 1 1 0 1 1 0 0 1 0]) = 5		5/30 = 17%
Fit([0 0 1 1 0 1 0 1 0 0]) = 4		4/30 = 13%
Fit([0 0 0 0 1 0 1 1 1 0]) = 4		4/30 = 13%
Fit([1 1 1 0 0 1 0 1 1 0]) = 4		4/30 = 13%
	30	

Select N individuals, with an individual being selected at each draw by the proportion of fitness it represents (often without replacement)

Fit([1 0 1 0 1 1 1 0 1 1]) = 7		7/30 = 23%
Fit([1 1 1 0 1 0 0 1 0 1]) = 6		6/30 = 20%
~~Fit([0 1 1 0 1 1 0 0 1 0]) = 5~~		5/30 = 17%
~~Fit([0 0 1 1 0 1 0 1 0 0]) = 4~~		4/30 = 13%
Fit([0 0 0 0 1 0 1 1 1 0]) = 4		4/30 = 13%
~~Fit([1 1 1 0 0 1 0 1 1 0]) = 4~~		4/30 = 13%
	30	

This is great for stochastically favoring better individuals more the higher their fitness is

But it comes with one major flaw too…

Suppose that later in optimization,
fitness values have gone up dramatically

Fit([1 0 1 0 1 1 1 0 1 1]) = 107
Fit([1 1 1 0 1 0 0 1 0 1]) = 106
Fit([0 1 1 0 1 1 0 0 1 0]) = 105
Fit([0 0 1 1 0 1 0 1 0 0]) = 104
Fit([0 0 0 0 1 0 1 1 1 0]) = 104
Fit([1 1 1 0 0 1 0 1 1 0]) = 104

630

Selection pressures is now much lower
(i.e. smaller chance of taking an uphill step) than
early on when average fitness values were smaller
(this is the opposite of what we want!)

Fit([1 0 1 0 1 1 1 0 1 1])	= 107	107/630=17.0%
Fit([1 1 1 0 1 0 0 1 0 1])	= 106	106/630=16.8%
Fit([0 1 1 0 1 1 0 0 1 0])	= 105	105/630=16.7%
Fit([0 0 1 1 0 1 0 1 0 0])	= 104	104/630=16.5%
Fit([0 0 0 0 1 0 1 1 1 0])	= 104	104/630=16.5%
Fit([1 1 1 0 0 1 0 1 1 0])	= 104	104/630=16.5%
	630	

Selection pressures is now much lower
(i.e. smaller chance of taking an uphill step) than
early on when average fitness values were smaller
(this is the opposite of what we want!)

Fit([1 0 1 0 1 1 1 0 1 1])	= 107	107/630=17.0%
Fit([1 1 1 0 1 0 0 1 0 1])	= 106	106/630=16.8%
Fit([0 1 1 0 1 1 0 0 1 0])	= 105	105/630=16.7%
Fit([0 0 1 1 0 1 0 1 0 0])	= 104	104/630=16.5%
Fit([0 0 0 0 1 0 1 1 1 0])	= 104	104/630=16.5%
Fit([1 1 1 0 0 1 0 1 1 0])	= 104	104/630=16.5%
	630	

Rank-based Selection

Define some selection proportion for each rank in the population *a priori*

e.g. exponential fall-off

1) 50%
2) 25%
3) 12.5%
4) 6.25%
5) 3.13%
6) 1.56%

We just apply are preset selection probabilities

Fit([1 0 1 0 1 1 1 0 1 1])	= 107	50%
Fit([1 1 1 0 1 0 0 1 0 1])	= 106	25%
Fit([0 1 1 0 1 1 0 0 1 0])	= 105	12.5%
Fit([0 0 1 1 0 1 0 1 0 0])	= 104	6.25%
Fit([0 0 0 0 1 0 1 1 1 0])	= 104	3.13%
Fit([1 1 1 0 0 1 0 1 1 0])	= 104	1.56%
	630	

And selection pressure is constant as average fitness changes

Fit([1 0 1 0 1 1 1 0 1 1]) = 7	50%
Fit([1 1 1 0 1 0 0 1 0 1]) = 6	25%
Fit([0 1 1 0 1 1 0 0 1 0]) = 5	12.5%
Fit([0 0 1 1 0 1 0 1 0 0]) = 4	6.25%
Fit([0 0 0 0 1 0 1 1 1 0]) = 4	3.13%
Fit([1 1 1 0 0 1 0 1 1 0]) = 4	1.56%
30	

Often we always want to include the most fit individual in the next population (“elitism”)

Fit([1 0 1 0 1 1 1 0 1 1]) = 7	100%
Fit([1 1 1 0 1 0 0 1 0 1]) = 6	25%
Fit([0 1 1 0 1 1 0 0 1 0]) = 5	12.5%
Fit([0 0 1 1 0 1 0 1 0 0]) = 4	6.25%
Fit([0 0 0 0 1 0 1 1 1 0]) = 4	3.13%
Fit([1 1 1 0 0 1 0 1 1 0]) = 4	1.56%
30	

Tournament Selection

Randomly select a subset (mini-batch) of your population, perform some other selection method (most frequently truncation selection) only on that subset and return the winners

Tournaments can be of any size

e.g. tournament of size 2

Fit([1 0 1 0 1 1 1 0 1 1]) = 7

→ Fit([1 1 1 0 1 0 0 1 0 1]) = 6

Fit([0 1 1 0 1 1 0 0 1 0]) = 5

→ Fit([0 0 1 1 0 1 0 1 0 0]) = 4

Fit([0 0 0 0 1 0 1 1 1 0]) = 4

Fit([1 1 1 0 0 1 0 1 1 0]) = 4

Tournaments can be of any size

e.g. tournament of size 2

Fit([1 0 1 0 1 1 1 0 1 1]) = 7

→ Fit([1 1 1 0 1 0 0 1 0 1]) = 6 ✓

Fit([0 1 1 0 1 1 0 0 1 0]) = 5

→ ~~Fit([0 0 1 1 0 1 0 1 0 0]) = 4~~

Fit([0 0 0 0 1 0 1 1 1 0]) = 4

Fit([1 1 1 0 0 1 0 1 1 0]) = 4

Usually you continue to select additional tournaments

→ Fit([1 0 1 0 1 1 1 0 1 1]) = 7 ✓

Fit([1 1 1 0 1 0 0 1 0 1]) = 6 ✓

→ ~~Fit([0 1 1 0 1 1 0 0 1 0]) = 5~~

~~Fit([0 0 1 1 0 1 0 1 0 0]) = 4~~

Fit([0 0 0 0 1 0 1 1 1 0]) = 4

Fit([1 1 1 0 0 1 0 1 1 0]) = 4

Usually you continue to select additional tournaments until all individuals have been evaluated and selected

Fit([1 0 1 0 1 1 1 0 1 1]) = 7 ✓

Fit([1 1 1 0 1 0 0 1 0 1]) = 6 ✓

~~Fit([0 1 1 0 1 1 0 0 1 0]) = 5~~

~~Fit([0 0 1 1 0 1 0 1 0 0]) = 4~~

→ Fit([0 0 0 0 1 0 1 1 1 0]) = 4 ✓

→ ~~Fit([1 1 1 0 0 1 0 1 1 0]) = 4~~

Small tournaments (e.g. size 2),
have low section pressure since
highly fit individuals can be knocked out

Fit([1 0 1 0 1 1 1 0 1 1]) = 7 ✓

~~Fit([1 1 1 0 1 0 0 1 0 1]) = 6~~

Fit([0 1 1 0 1 1 0 0 1 0]) = 5

Fit([0 0 1 1 0 1 0 1 0 0]) = 4

Fit([0 0 0 0 1 0 1 1 1 0]) = 4

Fit([1 1 1 0 0 1 0 1 1 0]) = 4

Small tournaments (e.g. size 2), have low section pressure since low fit individuals can survive

Fit([1 0 1 0 1 1 1 0 1 1]) = 7
Fit([1 1 1 0 1 0 0 1 0 1]) = 6
Fit([0 1 1 0 1 1 0 0 1 0]) = 5
Fit([0 0 1 1 0 1 0 1 0 0]) = 4
→ Fit([0 0 0 0 1 0 1 1 1 0]) = 4 ✓
→ ~~Fit([1 1 1 0 0 1 0 1 1 0]) = 4~~

At the other extreme, a tournament of the full populations is just regular truncation selection (and thus has maximum selection pressure)

- Fit([1 0 1 0 1 1 1 0 1 1]) = 7 ✓
- Fit([1 1 1 0 1 0 0 1 0 1]) = 6 ✓
- Fit([0 1 1 0 1 1 0 0 1 0]) = 5 ✓
- ~~Fit([0 0 1 1 0 1 0 1 0 0]) = 4~~
- ~~Fit([0 0 0 0 1 0 1 1 1 0]) = 4~~
- ~~Fit([1 1 1 0 0 1 0 1 1 0]) = 4~~

So tournament size is a great knob to use for controlling selection pressure

Tournaments are also great in that they can be used in “steady-state” algorithms, that perform variation-evaluation-selection loops on small subsets of the population at a time, instead of “generational” algorithms that have distinct iterations