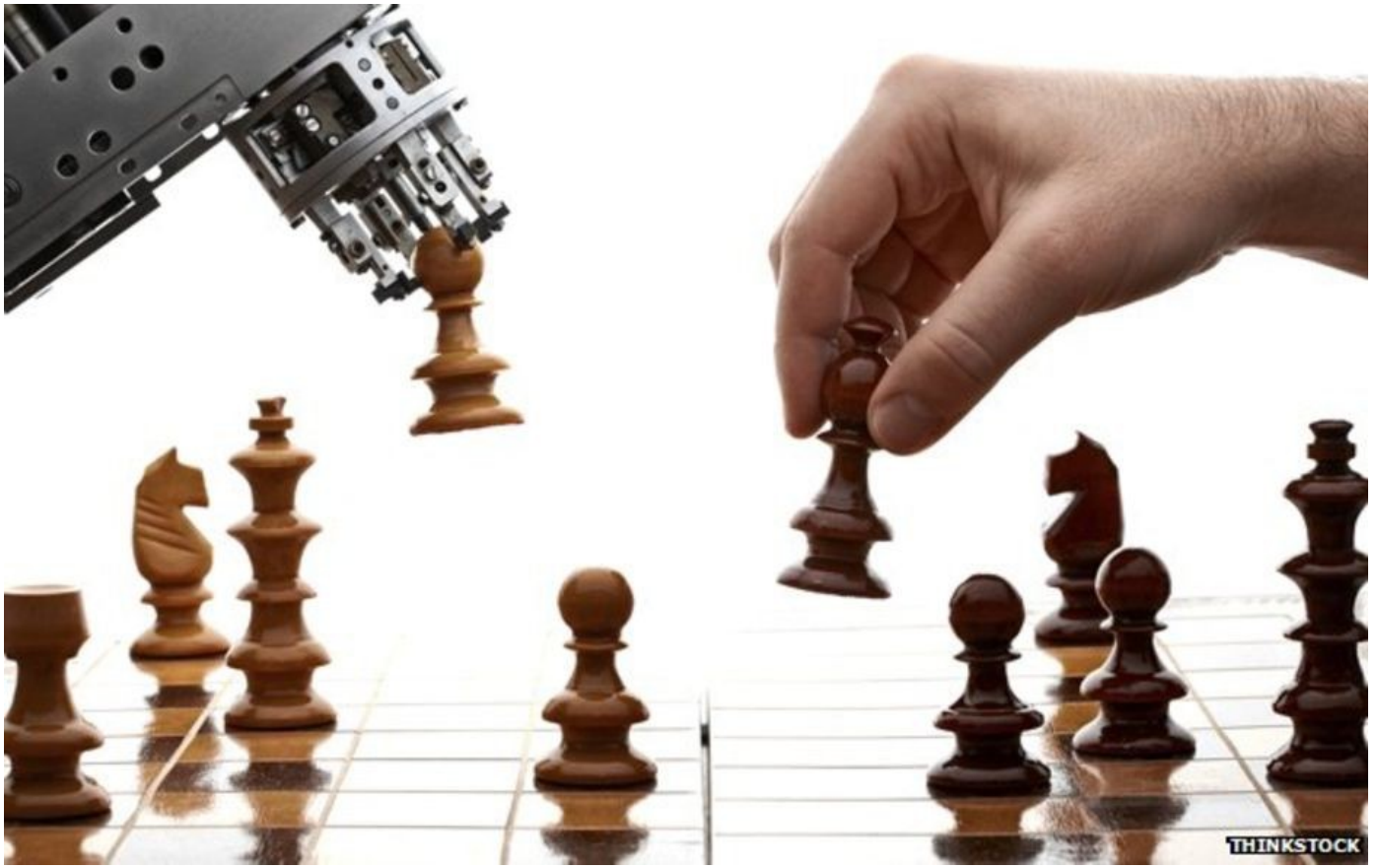


Introduction to Artificial Intelligence

COSC 4550 / COSC 5550

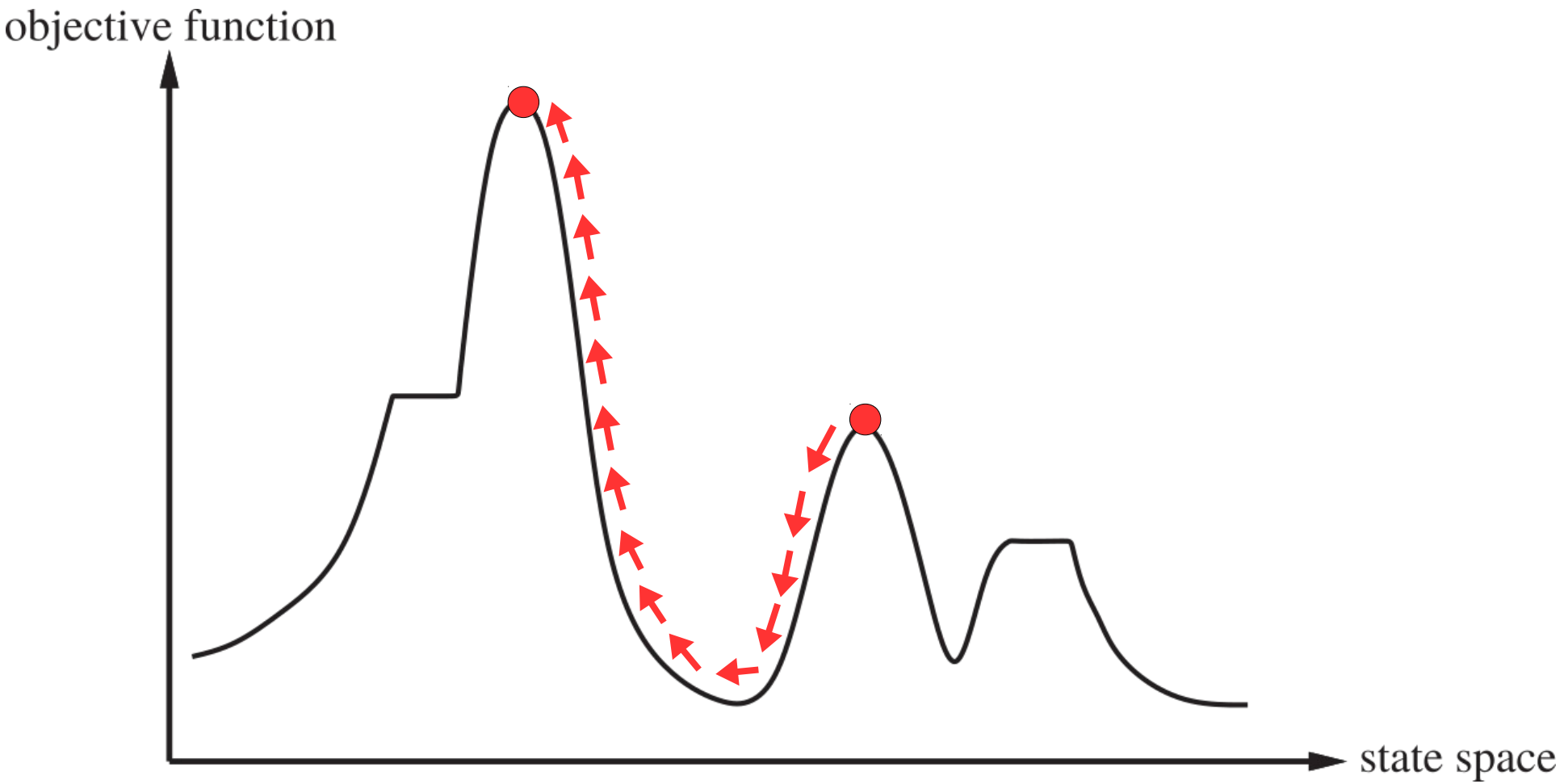
Professor Cheney

9/15/17

hillclimbing is greedy local search

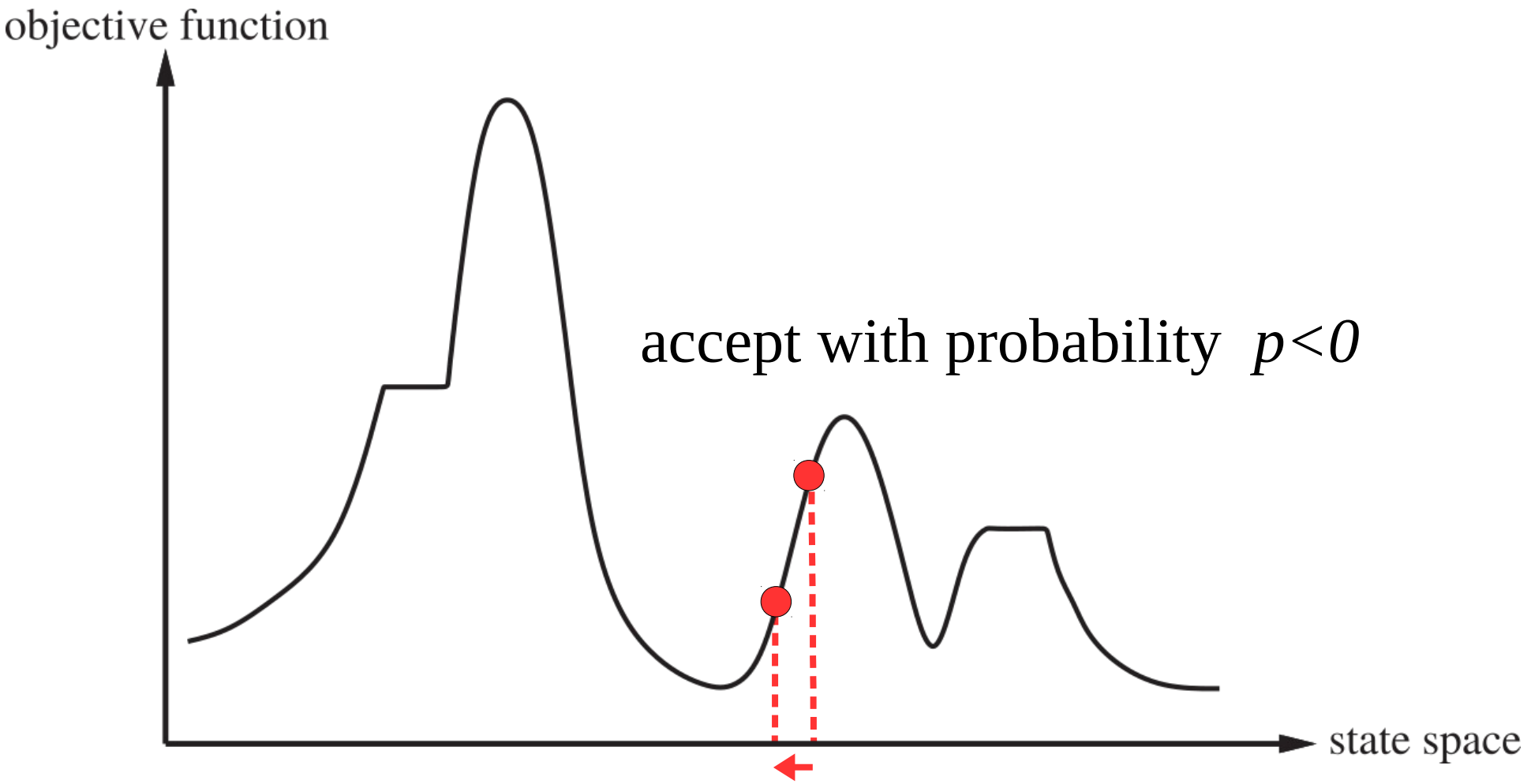
very simply to code and understand:

if your new state is better, keep it
if your new state is worse,
throw it away and keep your old state


objective function
state space

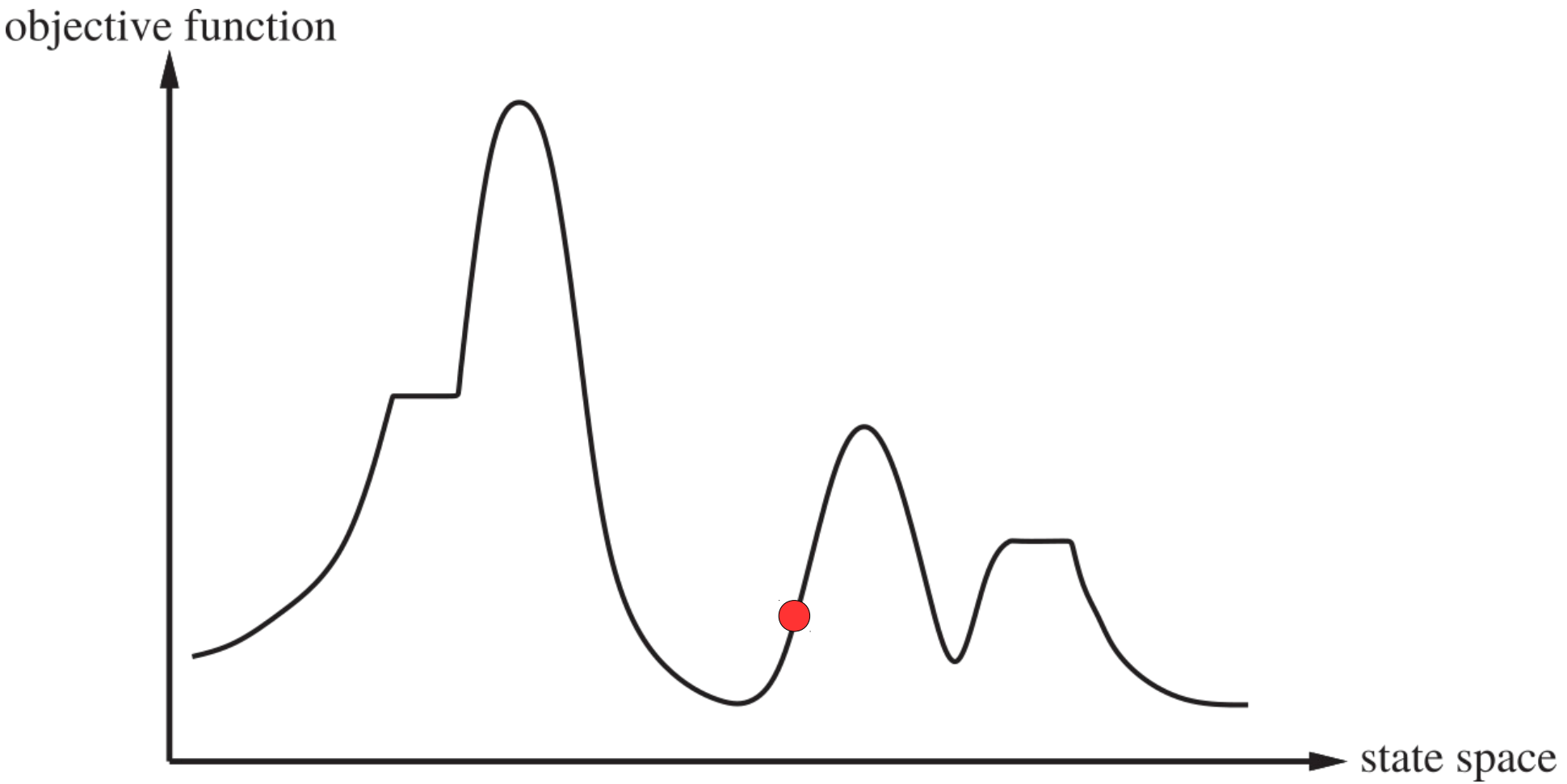
what if just sometimes,
we accepted negative mutations?

stochastic hillclimbing

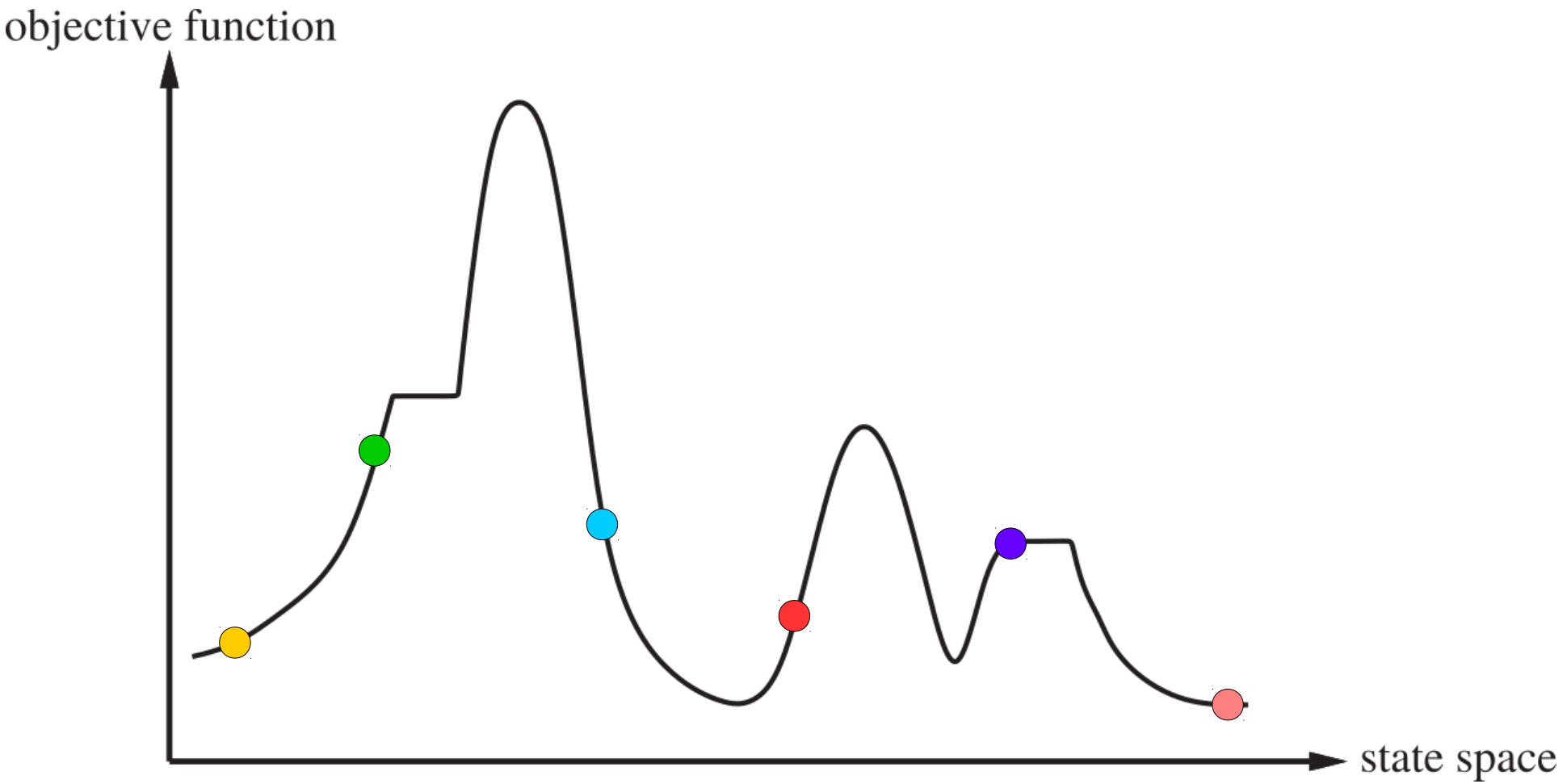


what if instead we used
random initialization?

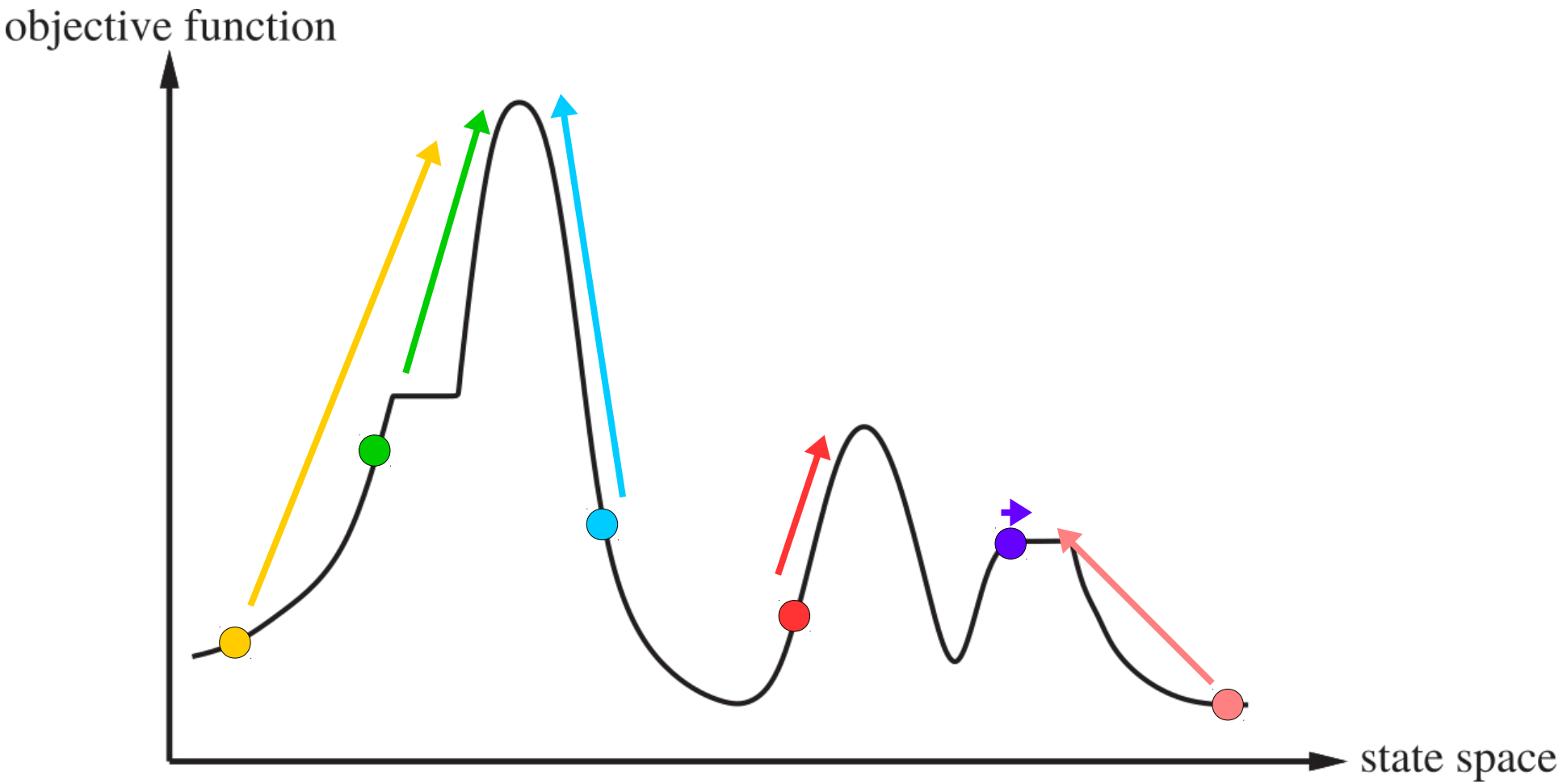
random-restart hillclimbing



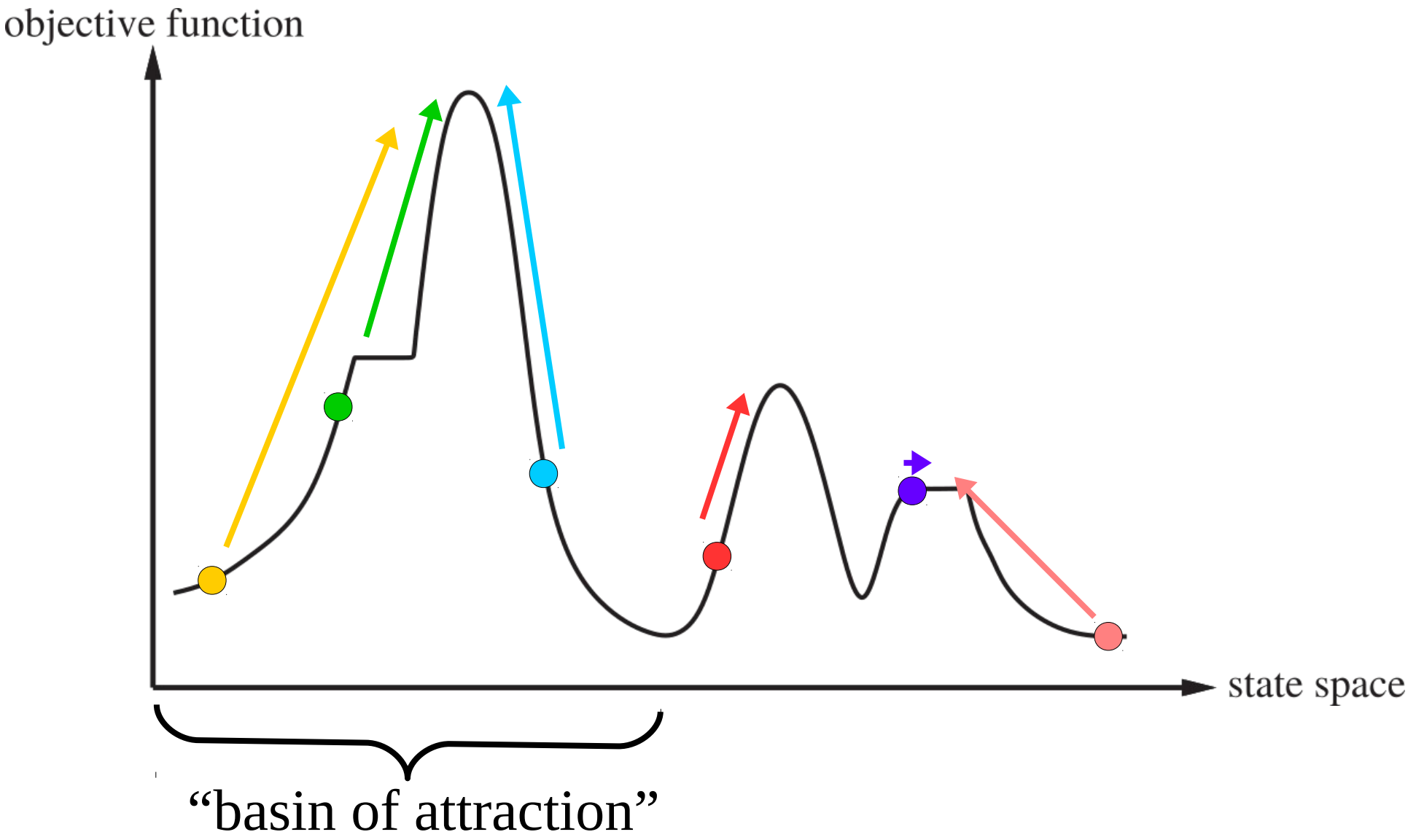
random-restart hillclimbing



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?)

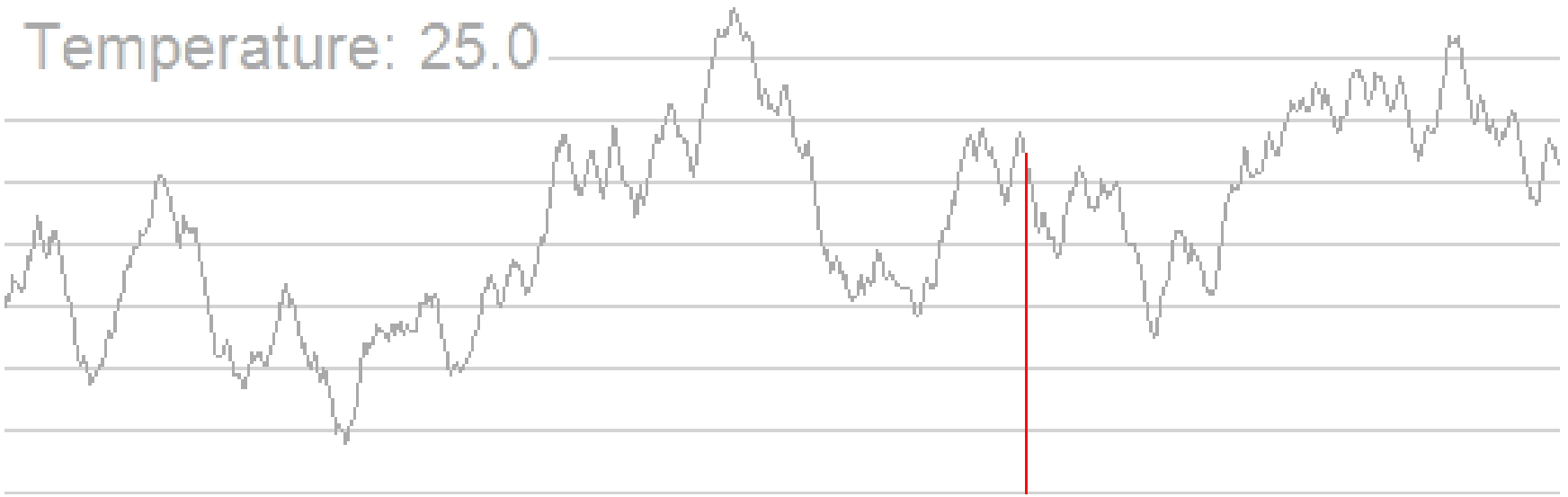
explore (take risky moves) early in search
so that you have more time to
catch up (and exploit) later on

simulated annealing

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

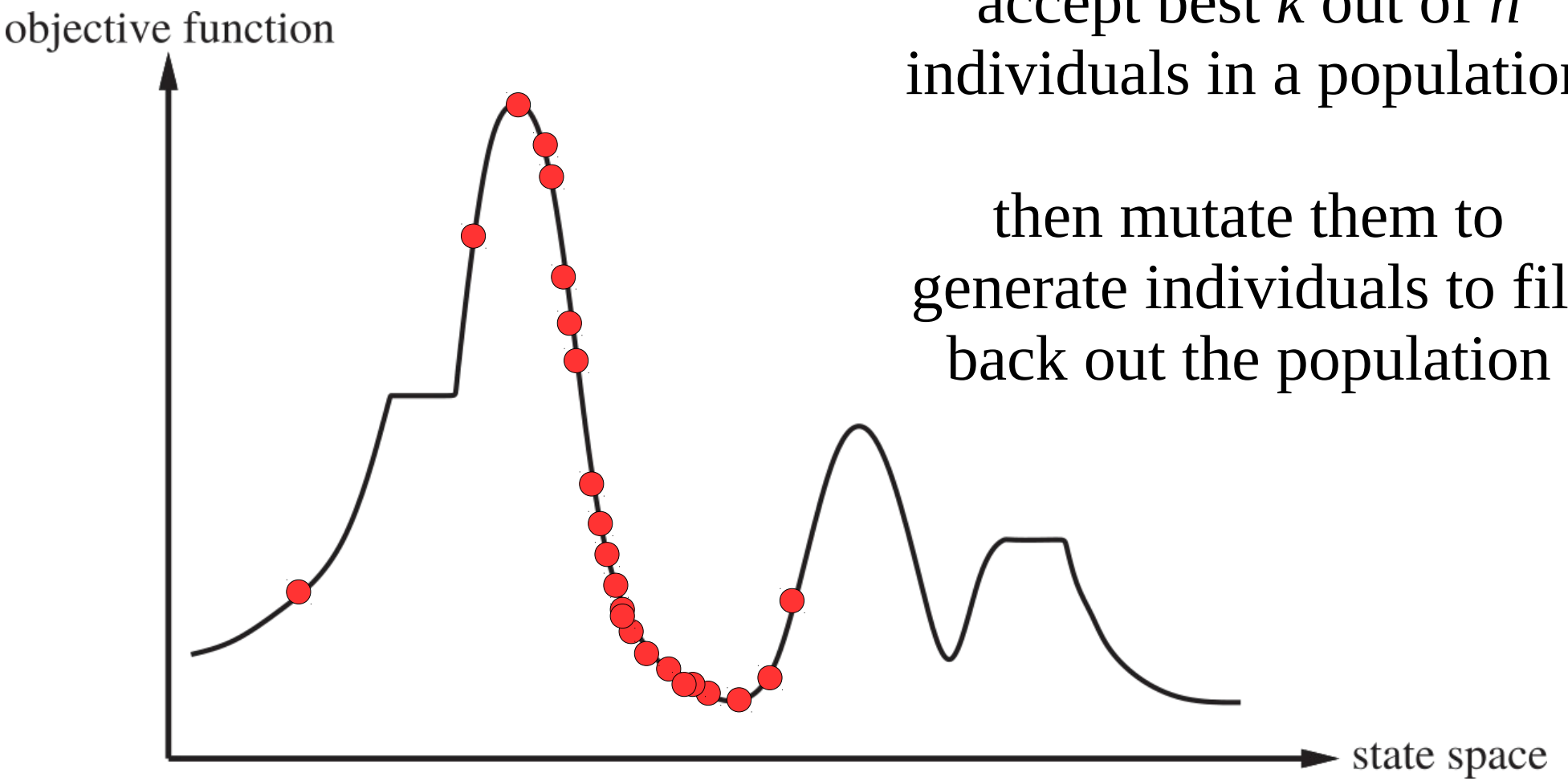
at first the point (current state) is randomly moving around all over the place (state space)

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

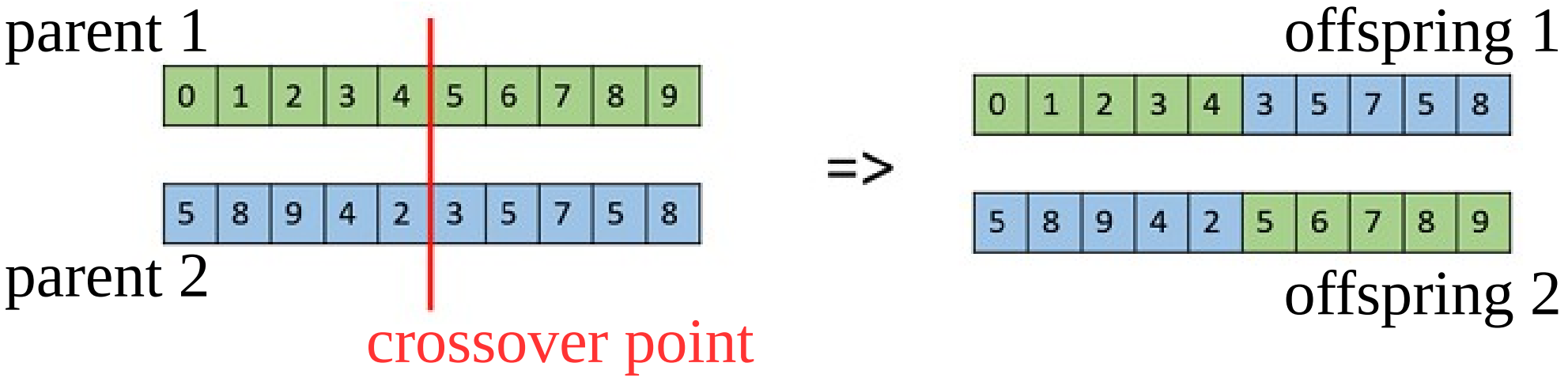


accept best *k* out of *n* individuals in a population

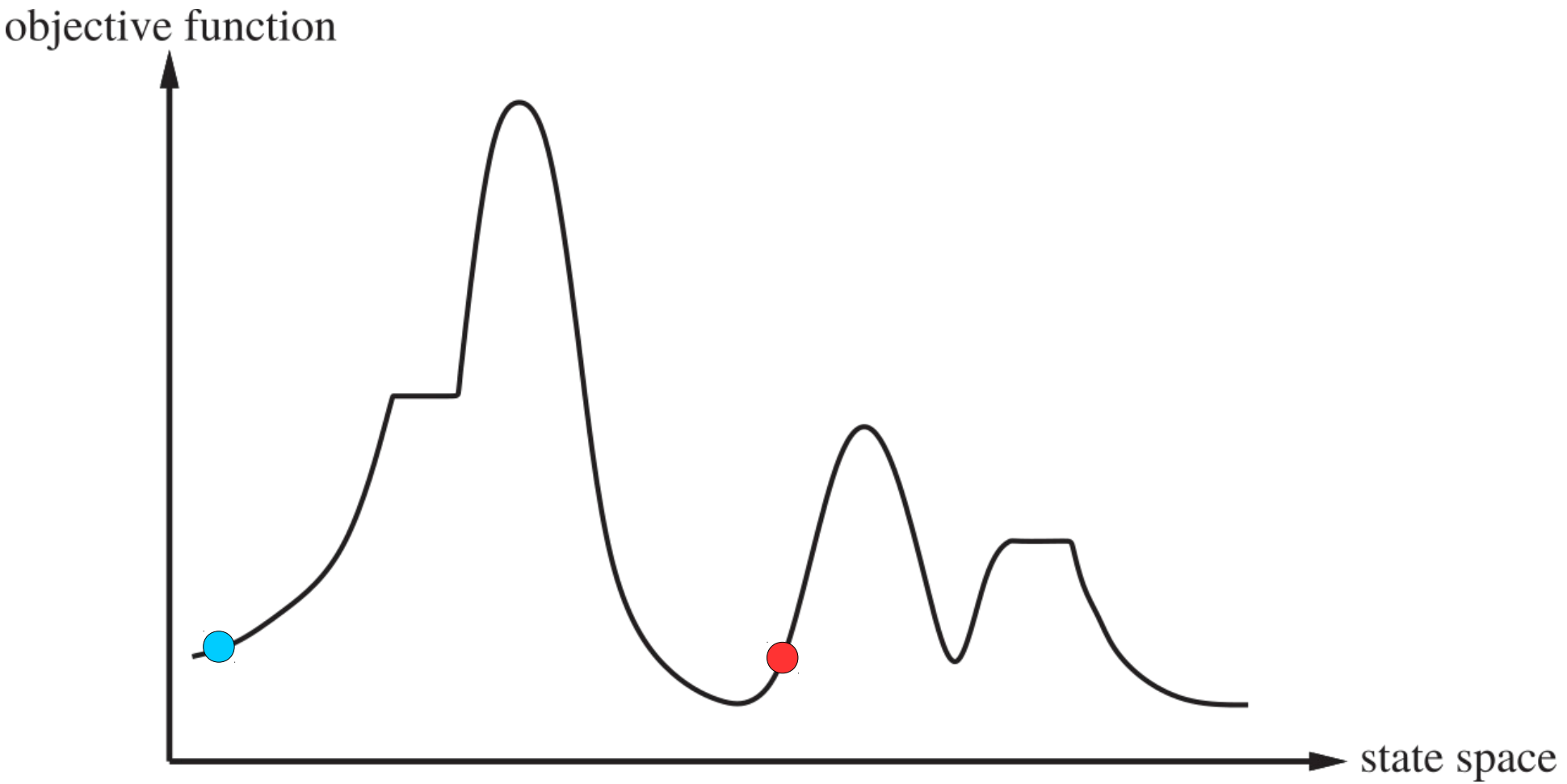
then mutate them to generate individuals to fill back out the population

making large random changes are unlikely to be beneficial

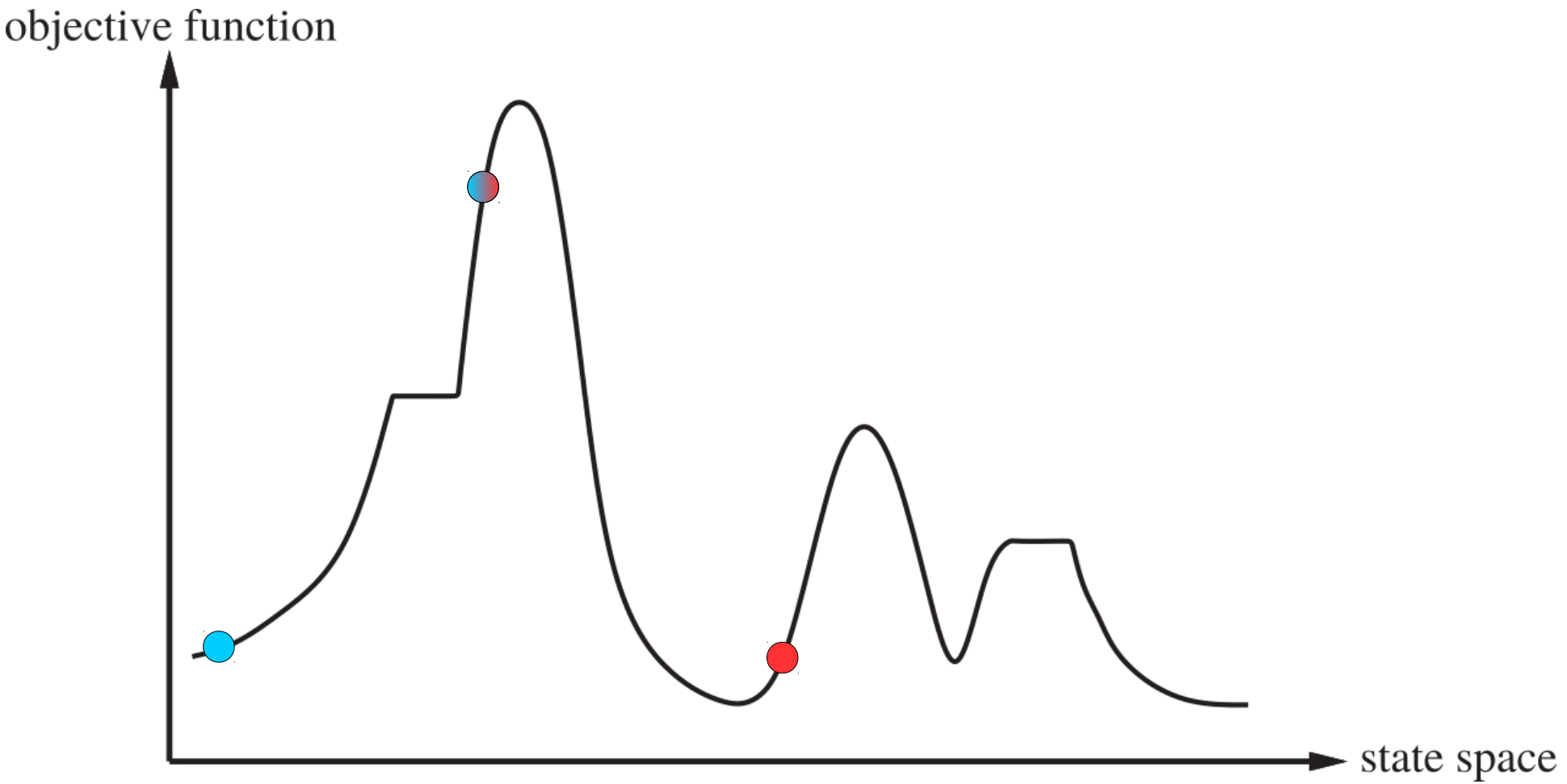
instead explore the space (make large changes) by combining (ideally the good) parts of multiple partial solutions



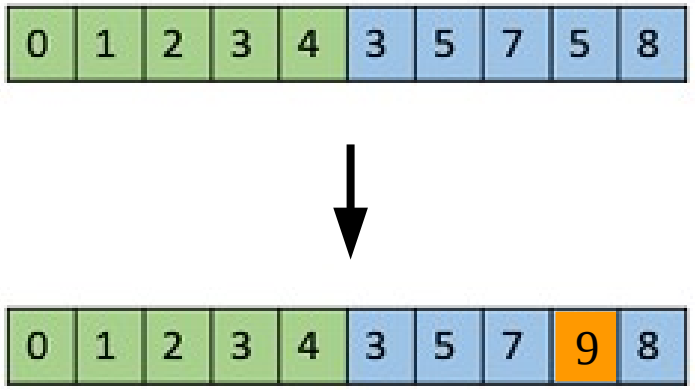
exploration through crossover

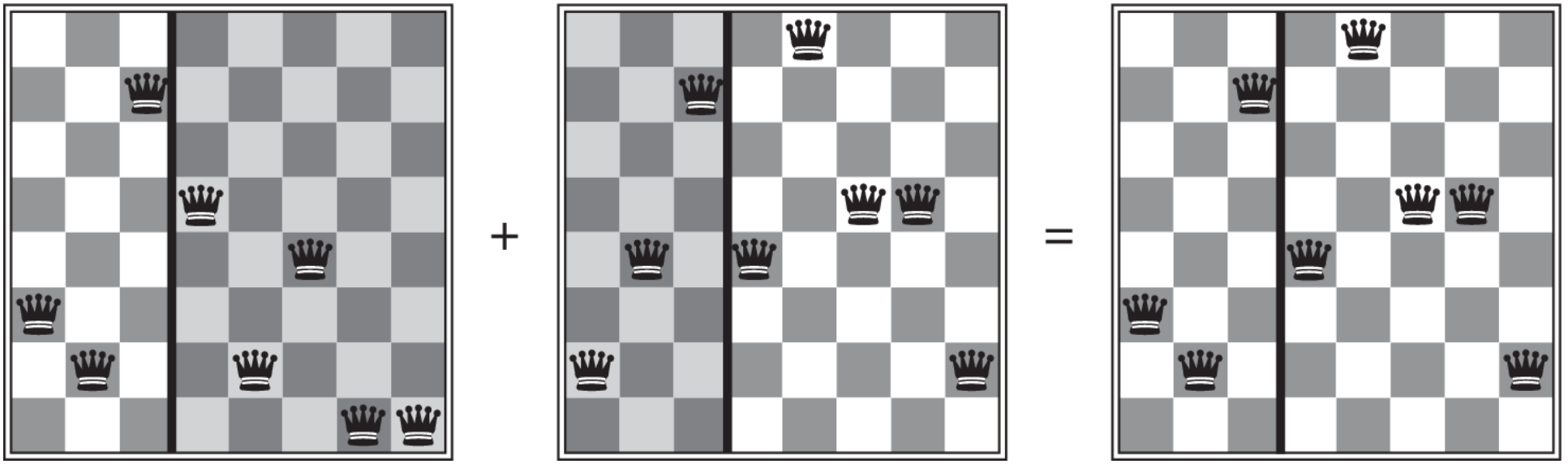


exploration through crossover



this may be combined with
single point mutations as well





+

=

???

Evolved Virtual Creatures

Karl Sims

Flexible Muscle-Based Locomotion for Bipedal Creatures

SIGGRAPH ASIA 2013

Thomas Geijtenbeek
Michiel van de Panne
Frank van der Stappen

to learn more, take:

COCS 4560 / COCS 5560
Modern Robots

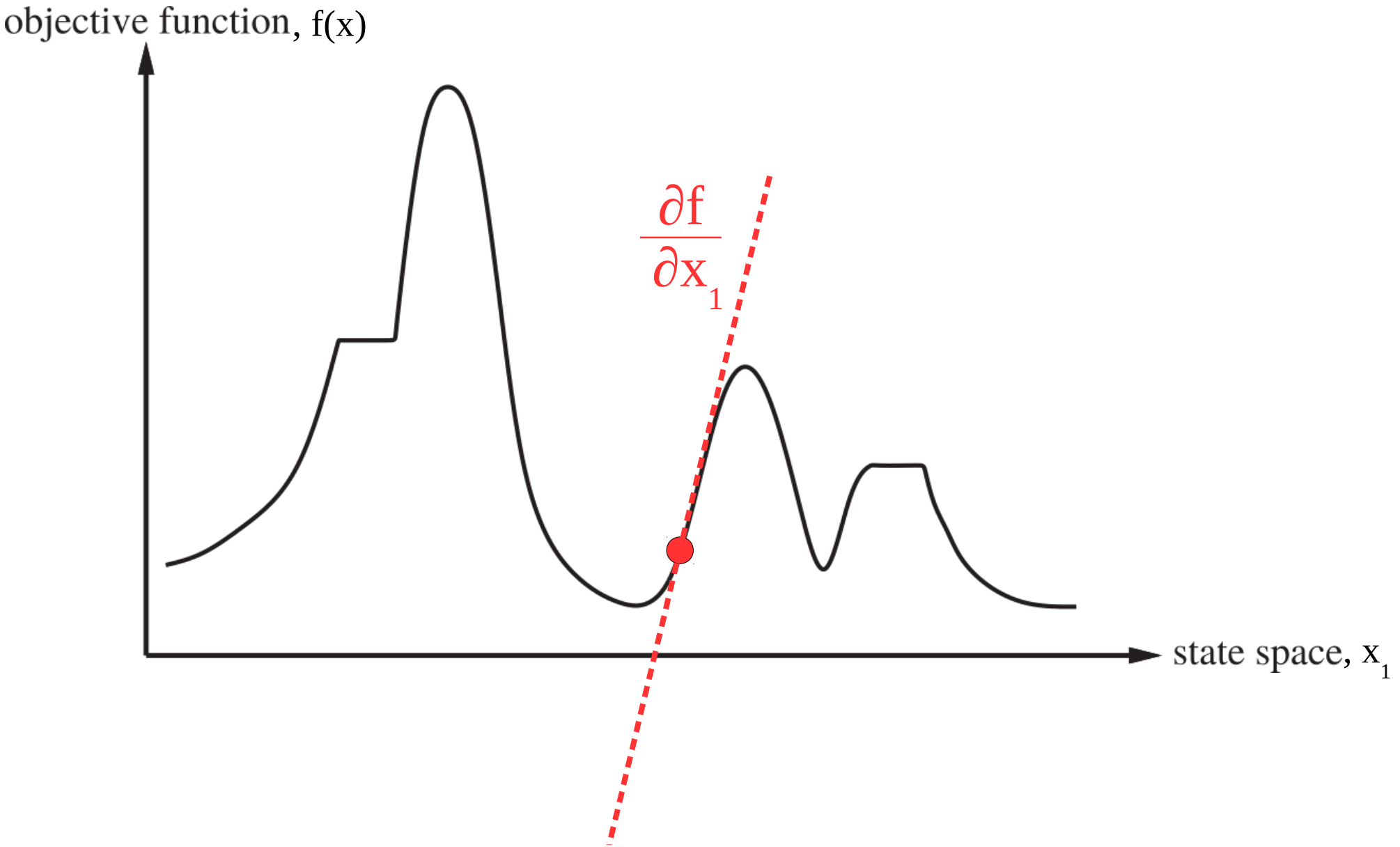
gradient-based methods

if we know our objective function, f(x),

(and it's differentiable)

we can just find the slope at our current point,

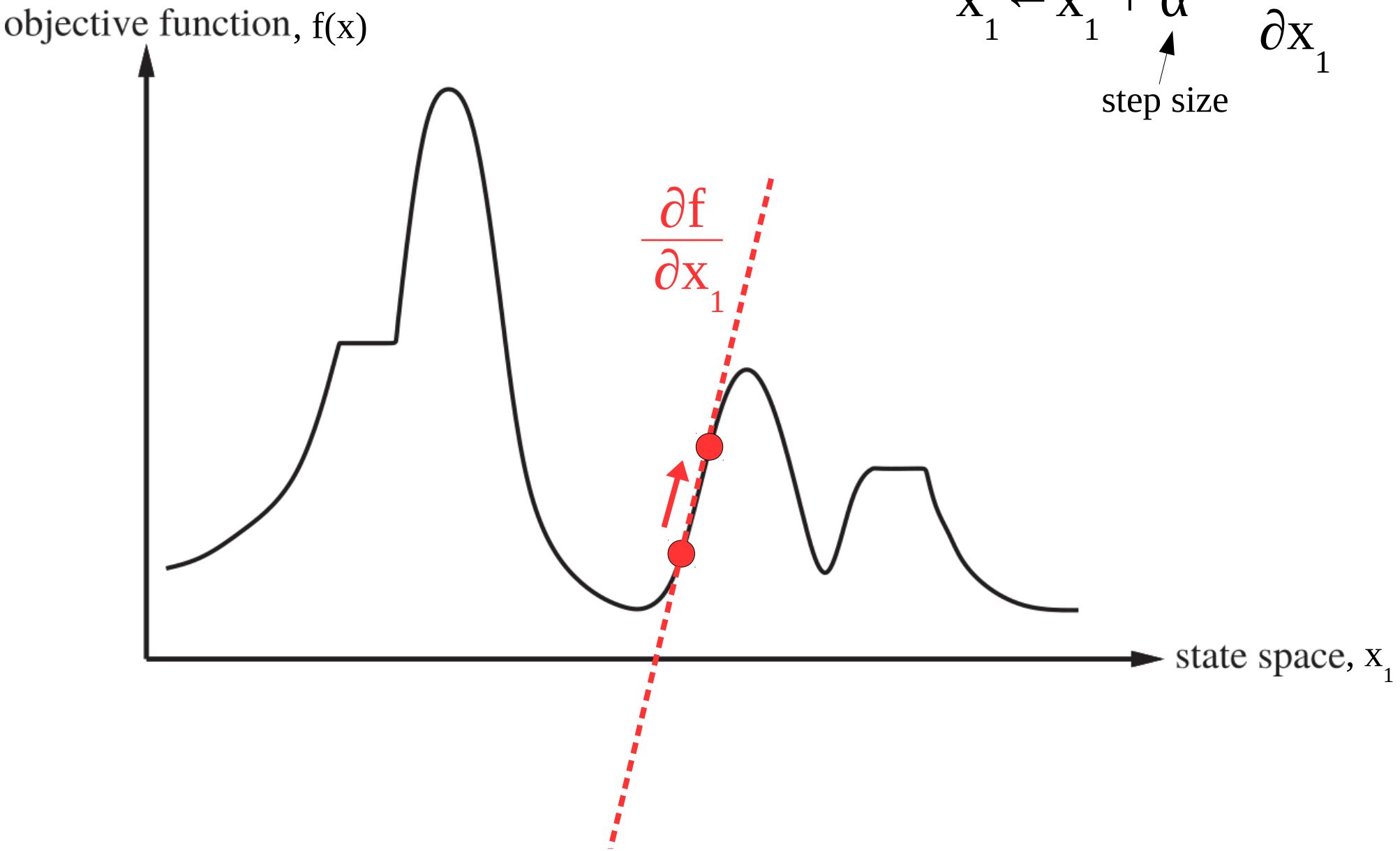
to tell us which direction to go to get better!


objective function, f(x)
$\frac{\partial f}{\partial x_1}$
state space, x_1

update x with:

$$x_1 \leftarrow x_1 + \alpha * \frac{\partial f}{\partial x_1}$$

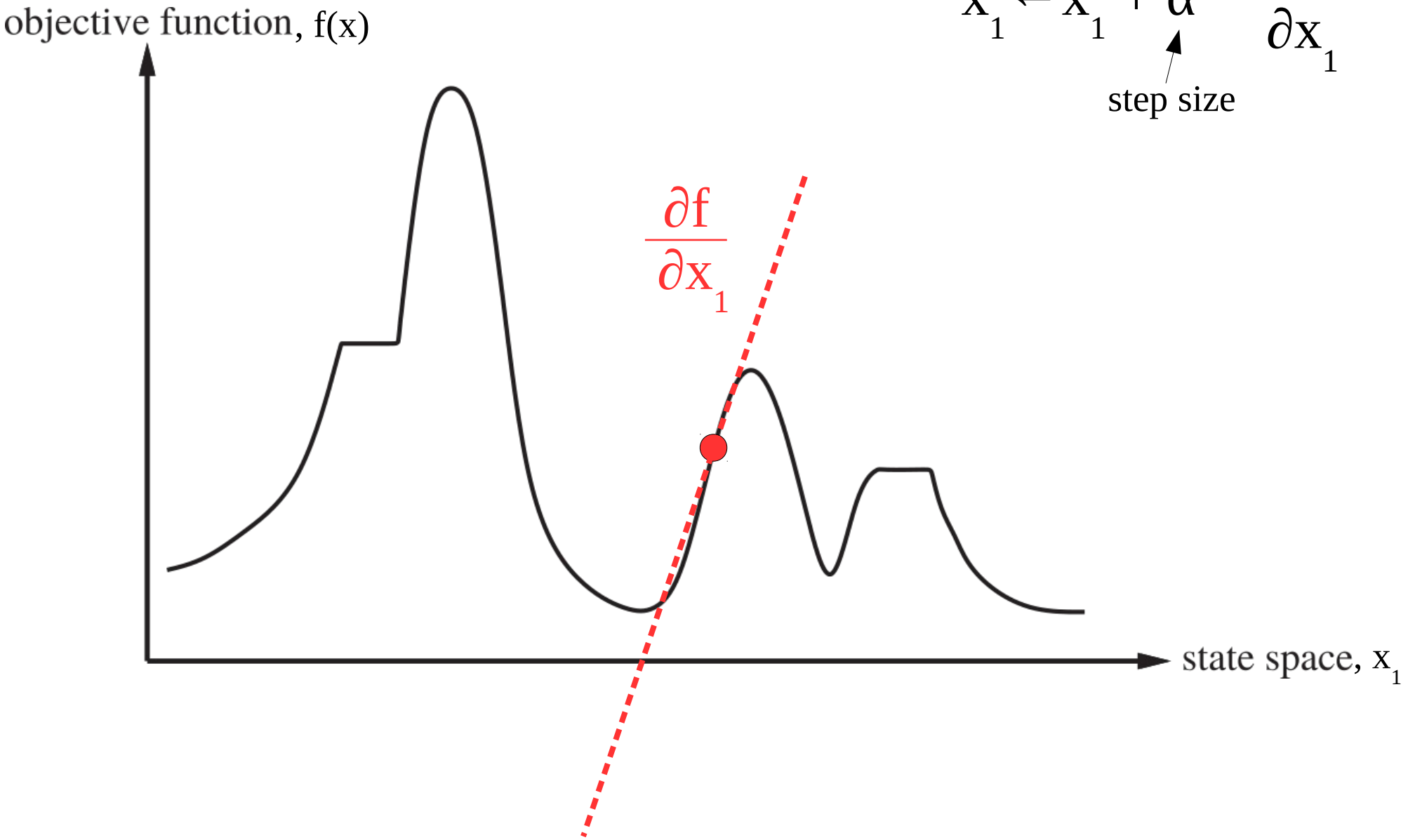
step size



update x with:

$$x_1 \leftarrow x_1 + \alpha * \frac{\partial f}{\partial x_1}$$

step size



more generally:
(for all parameters $x = \langle x_1, x_2, x_3, \ldots, x_n \rangle$)

$$x \leftarrow x + \alpha * \nabla f$$

$$\nabla f = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \ldots, \frac{\partial f}{\partial x_n} \right\rangle$$

"gradient" of f(x)

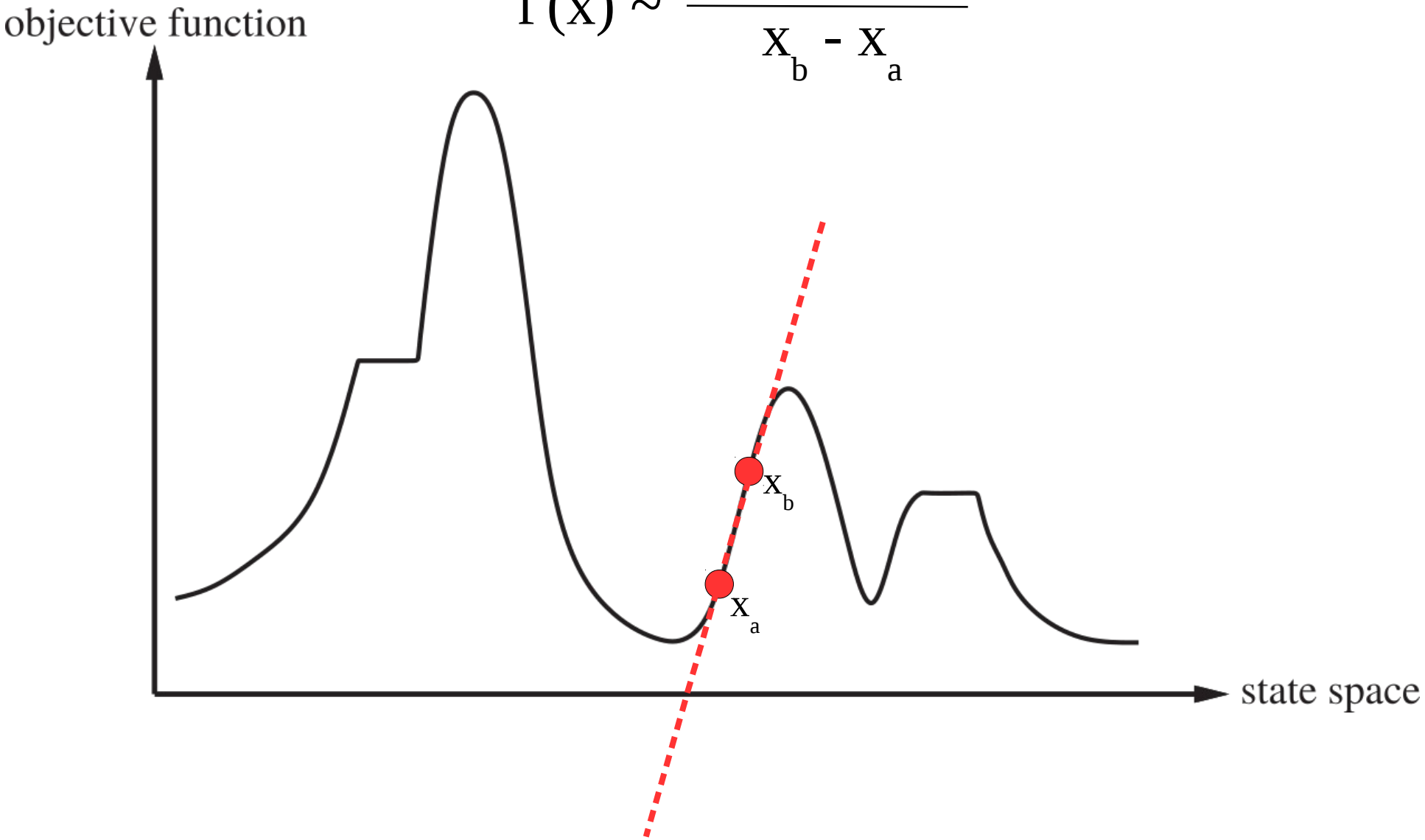
this means, for each x_i, consider the effect of changing (only) x_i on the value of f(x)

if you have two nearby points,
you can estimate the derivative of f(x) as:

$$f'(x) \sim \frac{f(x_b) - f(x_a)}{x_b - x_a}$$

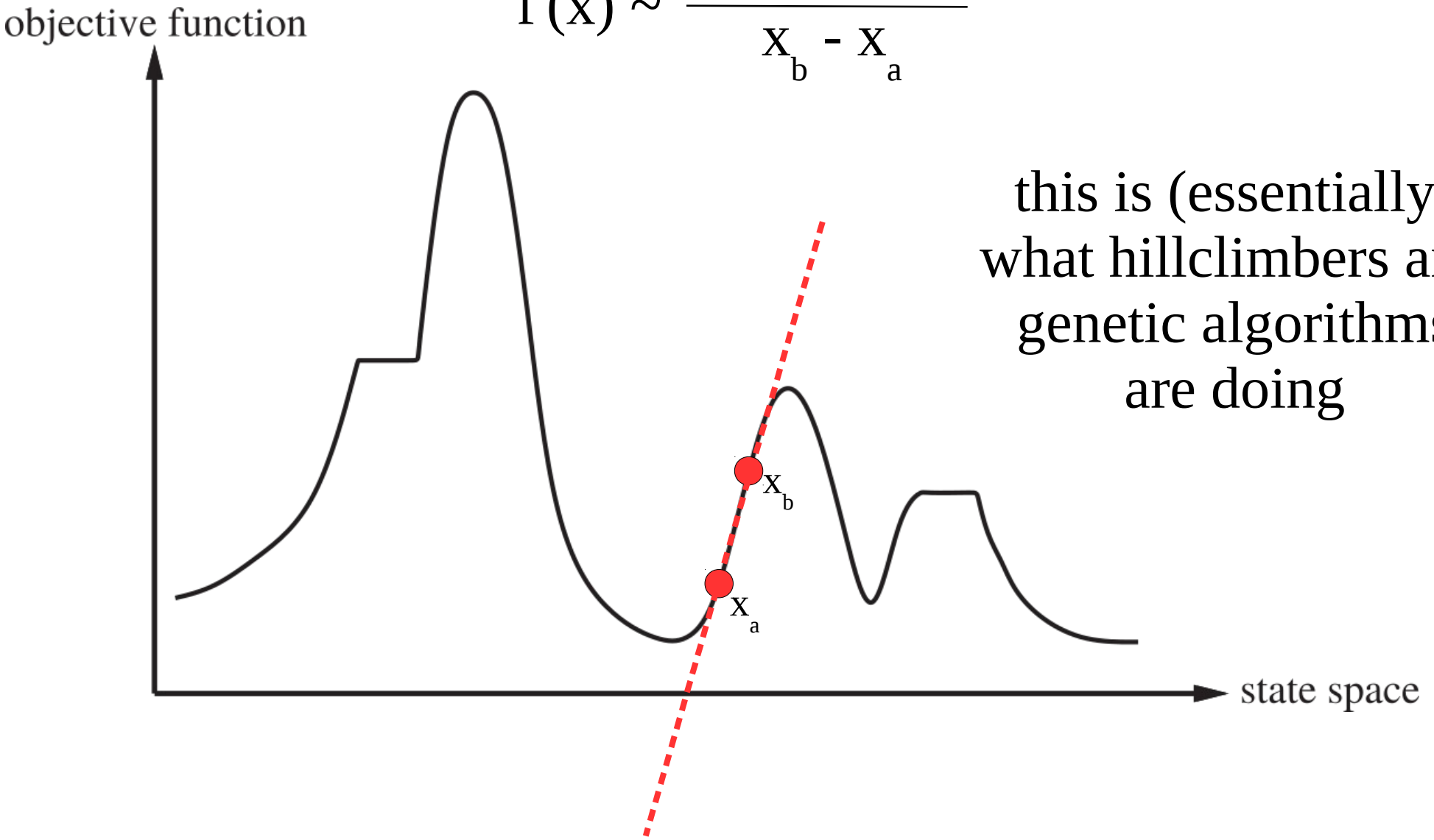
if you have two nearby points,
you can estimate the derivative of f(x) as:

$$f'(x) \sim \frac{f(x_b) - f(x_a)}{x_b - x_a}$$



if you have two nearby points,
you can estimate the derivative of f(x) as:

$$f'(x) \sim \frac{f(x_b) - f(x_a)}{x_b - x_a}$$



this is (essentially)
what hillclimbers and
genetic algorithms
are doing

to learn more,
take numerical methods:

COCS 3340 – Intro to Scientific Computing

COCS 5310/5340/5345 – Computational Methods in Applied Sciences (I,II,III)

COCS 4340 – Numerical Methods for Ordinary and Partial Differential Equations