Communications Network Design

lecture 14

Matthew Roughan
<matthew.roughan@adelaide.edu.au>

Discipline of Applied Mathematics
School of Mathematical Sciences
University of Adelaide

May 14, 2009

Randomized algorithms: simulated annealing

It is often the case that we optimize against a non-convex objective function. In these cases we often use heuristics such as gradient descent, but they can become stuck in a local minimum. **Simulated annealing** allows our search to “bounce” out of such a point, by including some randomization in its search. We present here the **Metropolis** algorithm for simulated annealing.

Star-like networks

- earlier, we considered designing a hub-spoke (star-like) network
  - cost based on link length
  - equivalent to $\beta_e \propto d_e$ and, $\alpha_e = 0$
  - as before (e.g. for Prim), this is only construction costs
  - can we include a load based cost α_e?
- design a star where the costs will be

$$C(\mathbf{f}) = \sum_{e \in T} \alpha_e f_e$$

  - set $\beta_e = 0$ this time

Star-like networks

- approach: simple case $\alpha_e = 1$
  - find the hub node which maximizes the flows which go-to, or leave from the star, i.e.,
$$\text{hub} = \operatorname*{argmin}_{p \in N} \left\{ \sum_{q \in N} t_{pq} \right\}$$
  - this minimizes the traffic which has to take two hops
  - we can consider all $|N|$ possibilities in $O(|N|)$ time, with $O(|N|)$ operations per case, so $O(|N|^2)$
- generalizes to $\alpha_e \neq const$, by finding the hub node
$$\text{hub} = \operatorname*{argmin}_{h \in N} \sum_{p \in N} \alpha_{ph} \sum_{q \in N} t_{pq}$$

Star-like networks

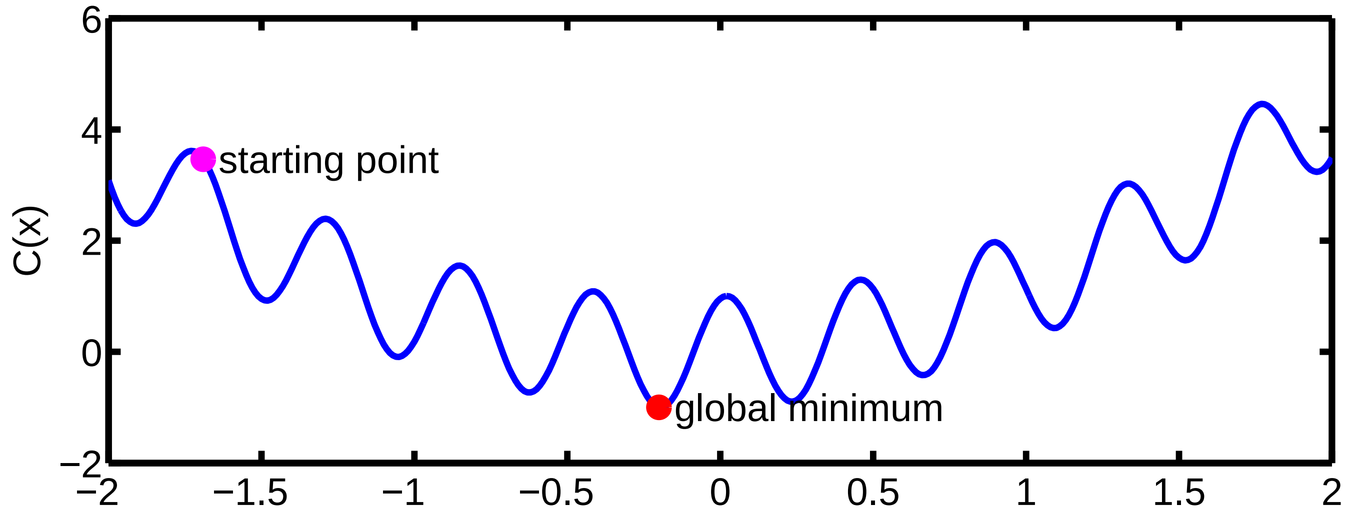
- no-one designs star-like networks like this
  - they do use stars, but not designed as above
  - e.g. WAN
    - when we decide the "hub", we put all of our servers there (e.g. web and email servers)
    - most traffic in enterprise WANs is local, or from client to server
    - if the servers are put somewhere, the traffic will go there anyway
    - so the traffic pattern depends on our design!
  - Broadcast network
    - traffic all originates at the hub
- for more complex (better) designs, the problem is NP-hard

Some problems are too hard

- some problems are two big to solve
  - even polynomial time algorithms can run out of puff
  - NP-hard problems are a problem
- rounding errors in computations
  - lead to incorrect or meaningless solutions
  - ill-posedness
- sometimes we can't write down the cost
  - "I don't know much about art, but I know what I like"
  - we can work out the cost for a solution, but we don't know what the cost function looks like
  - hence we can't exploit problem specifics

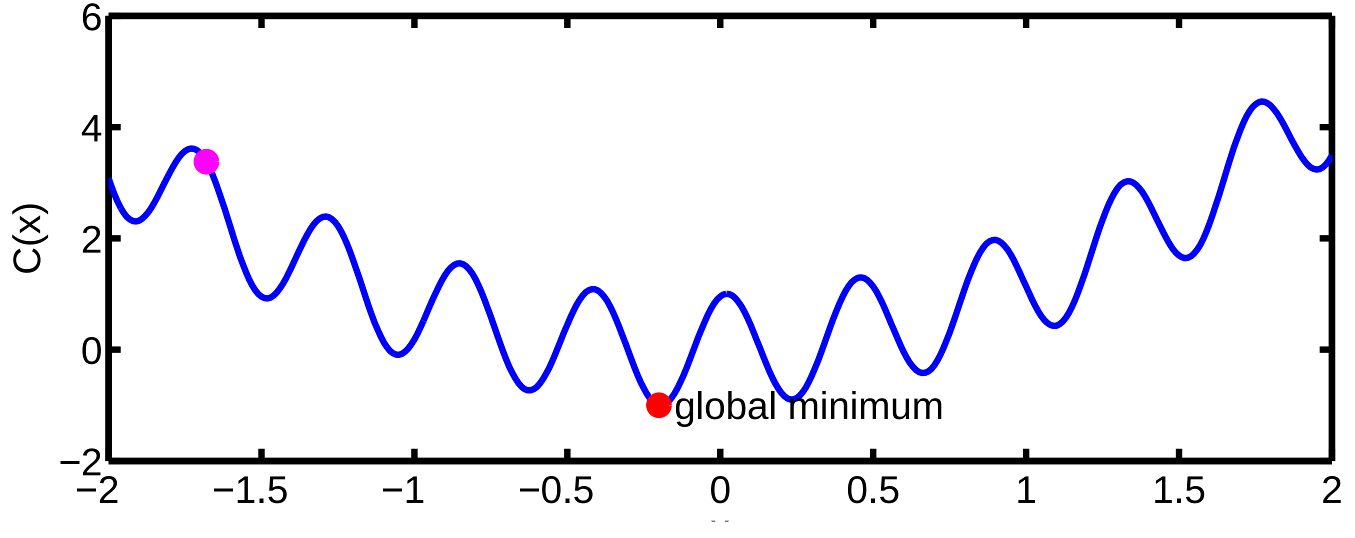
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



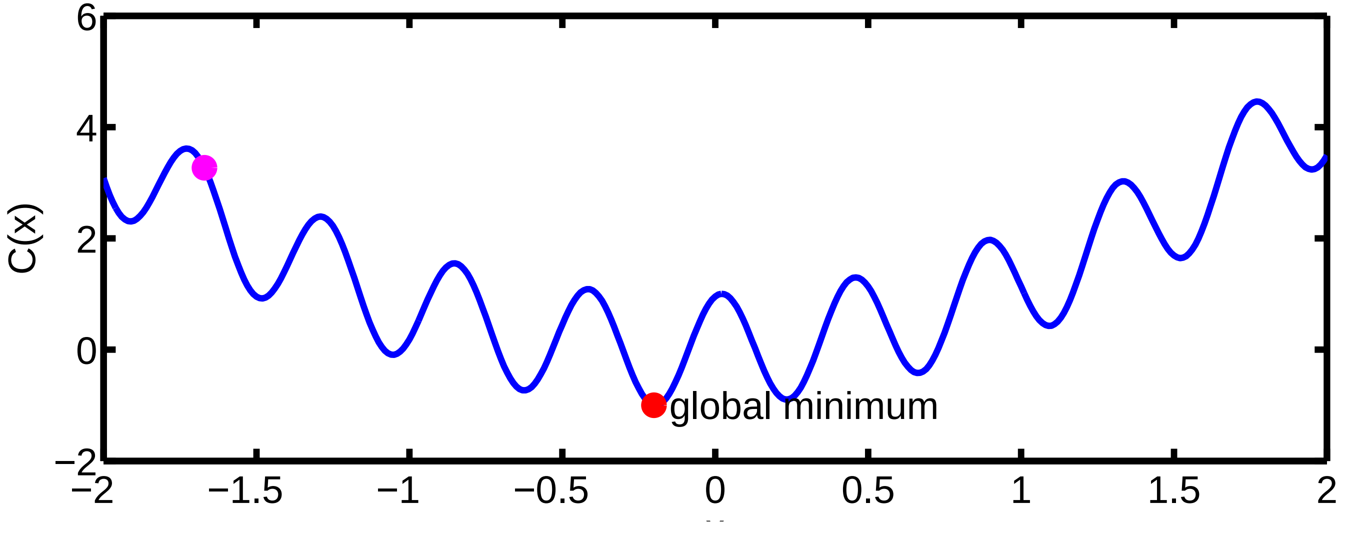
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



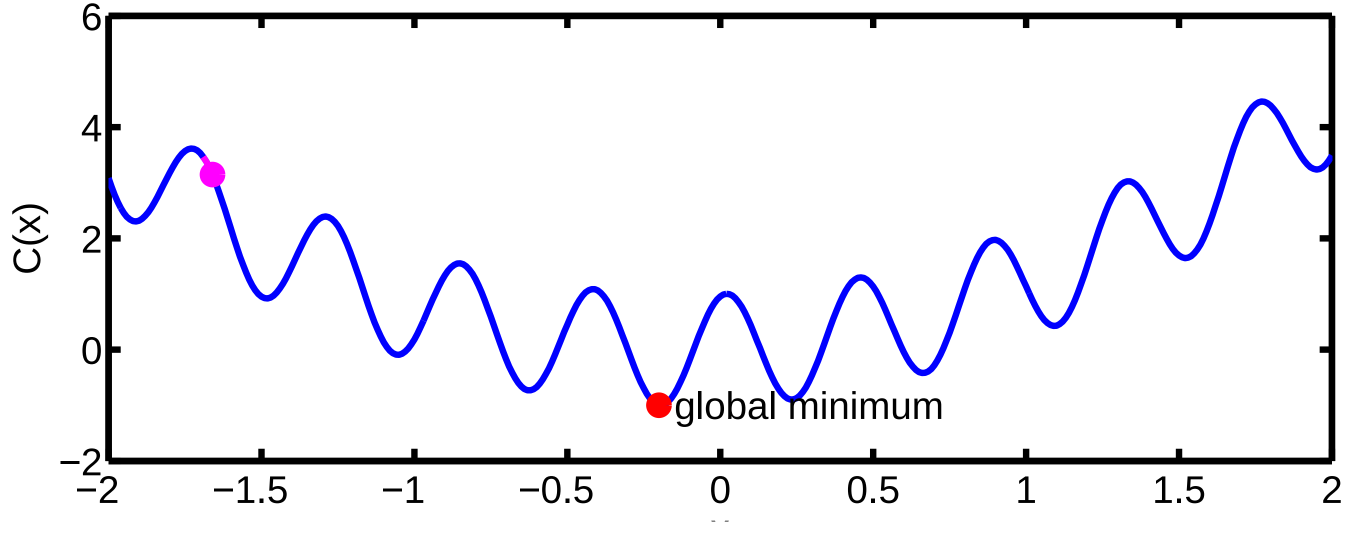
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



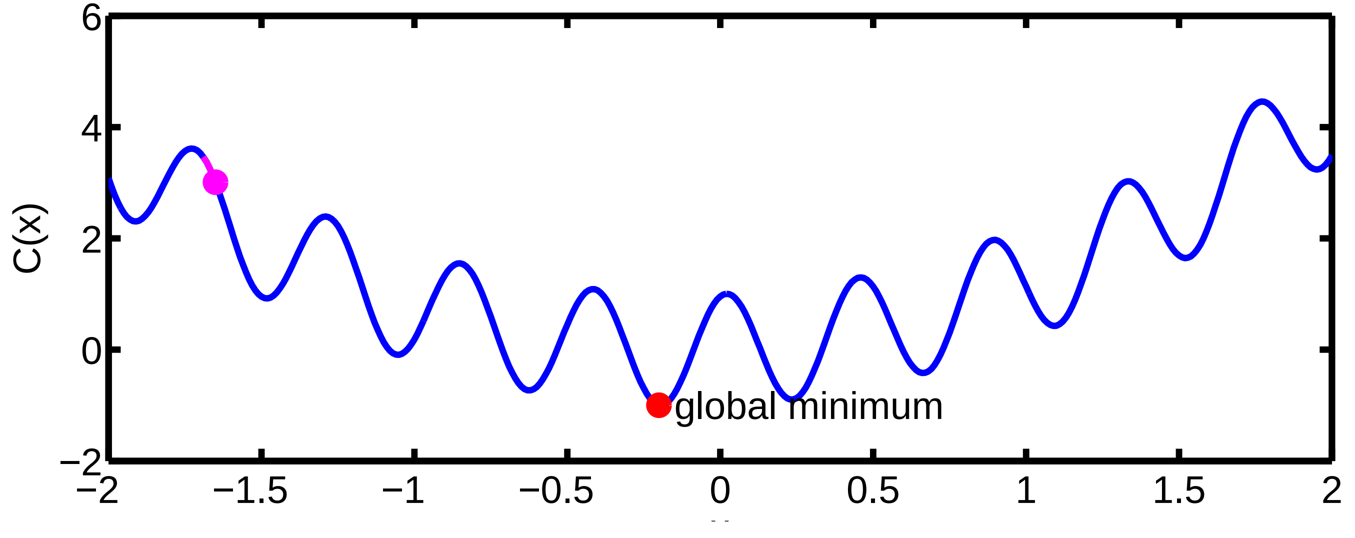
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



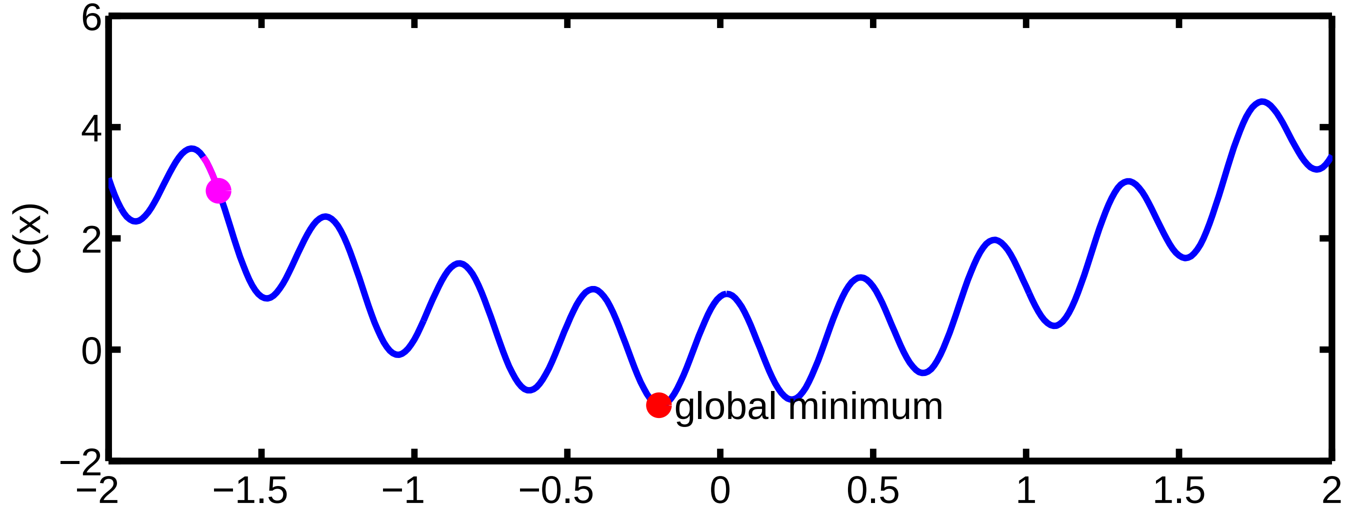
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



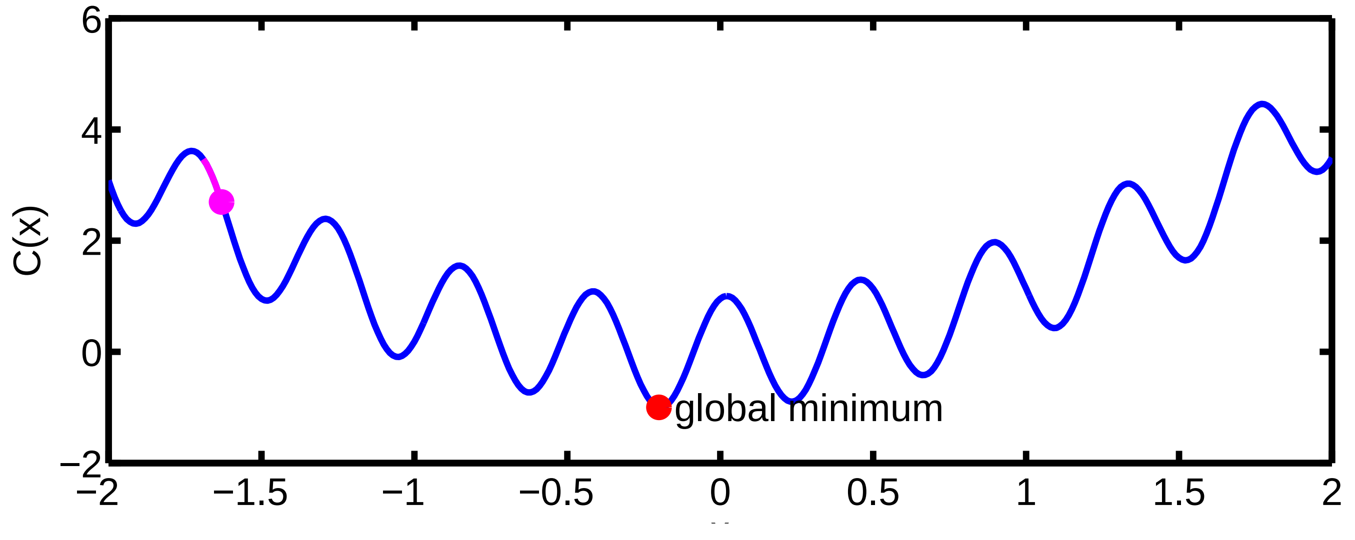
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



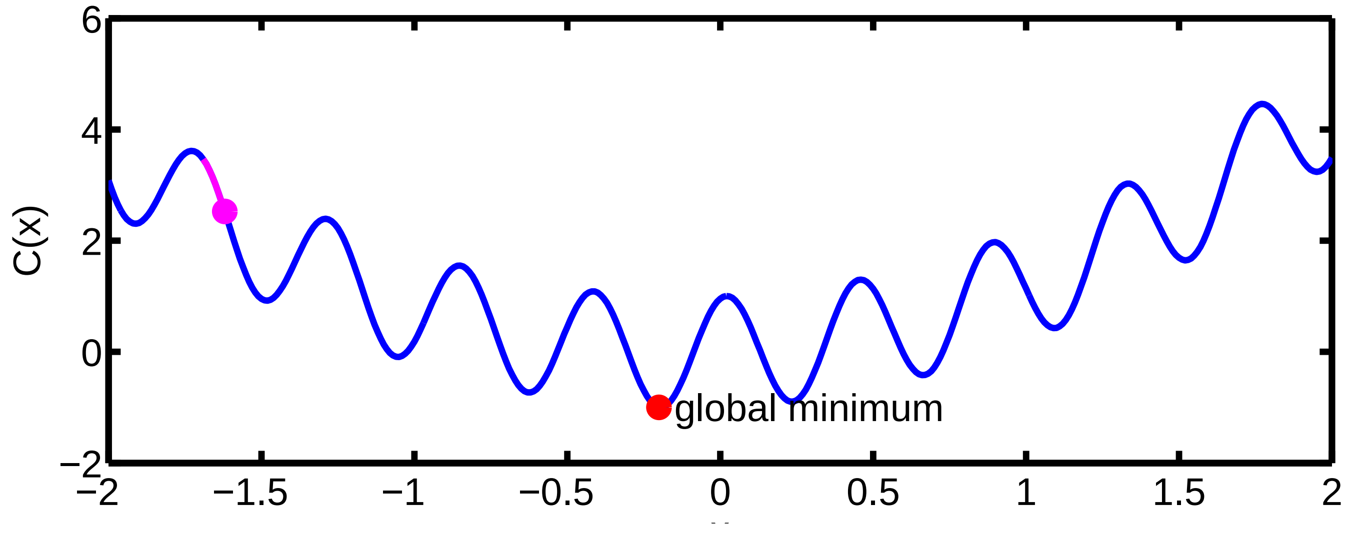
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



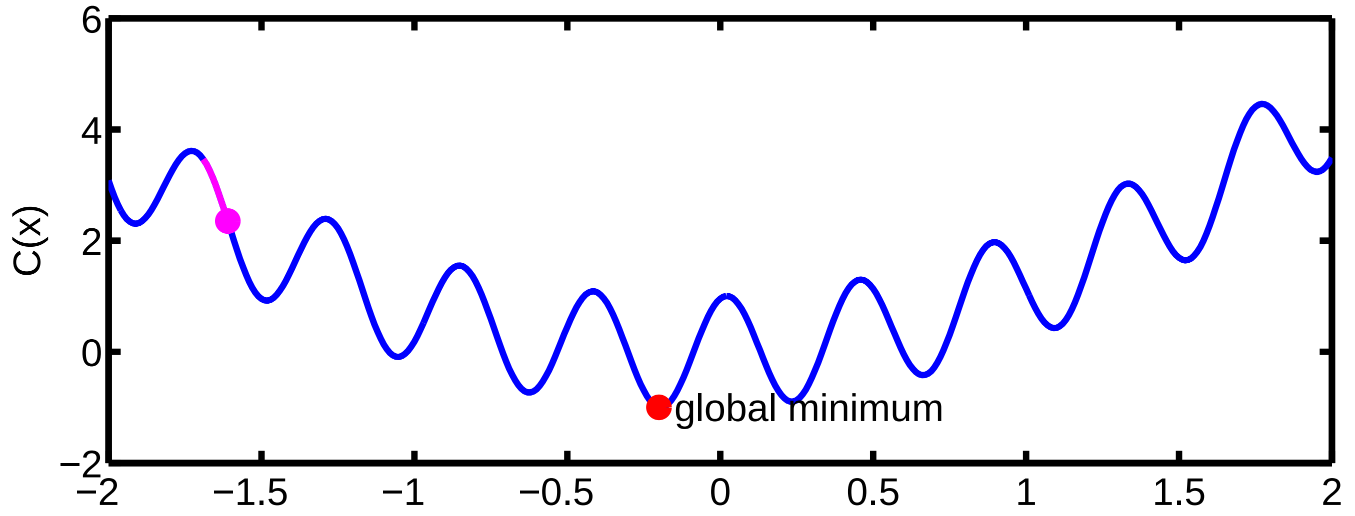
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



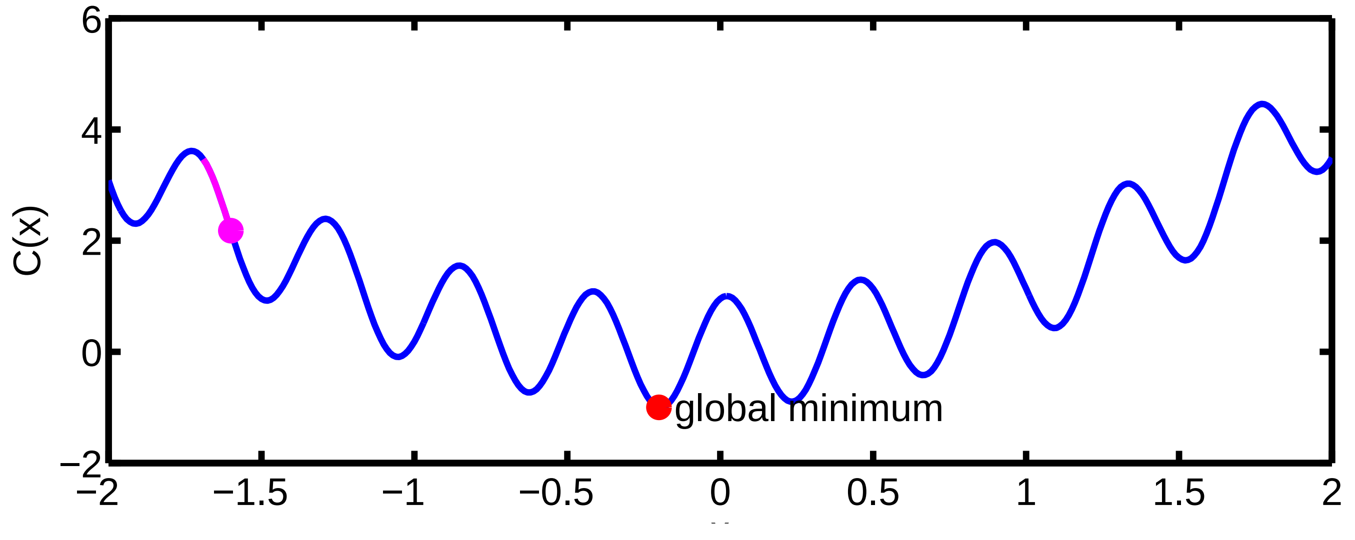
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



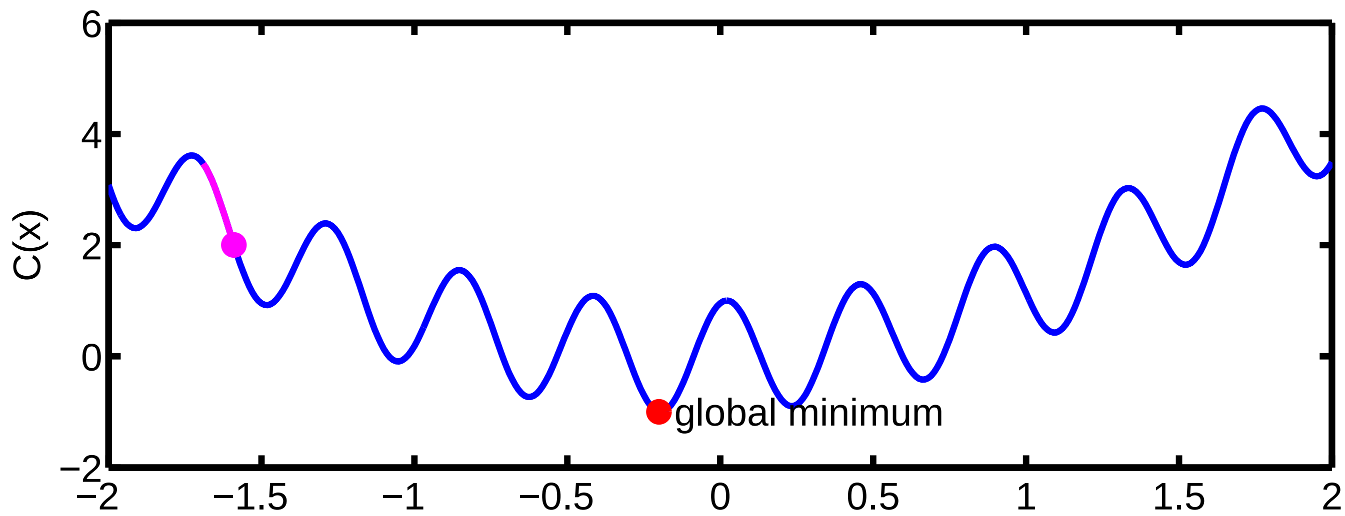
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



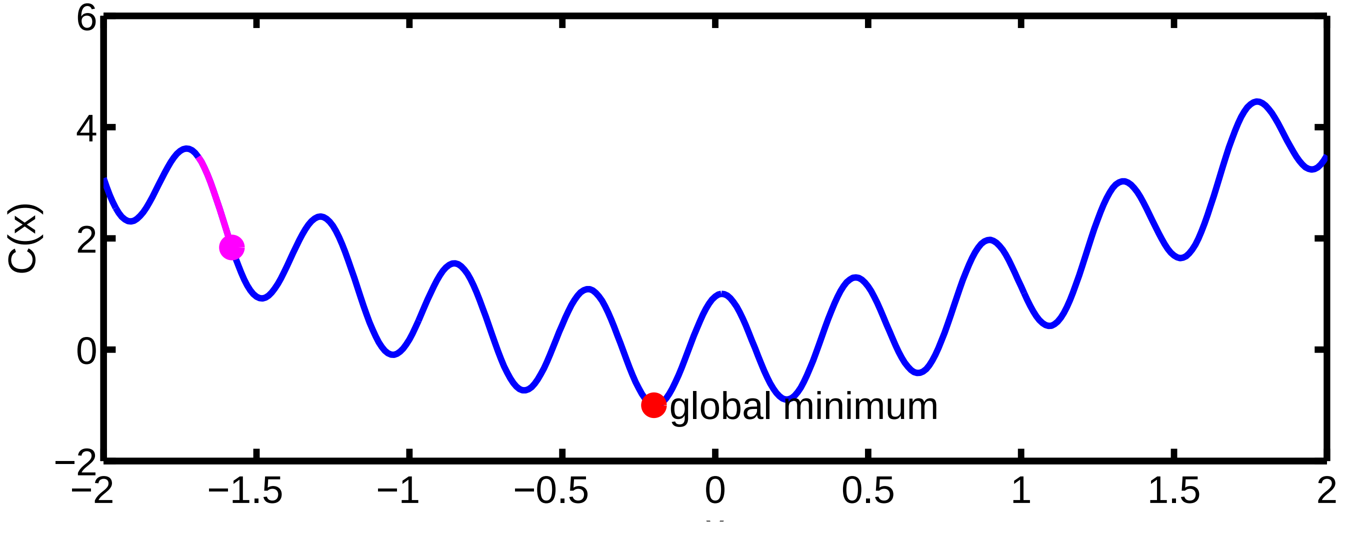
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



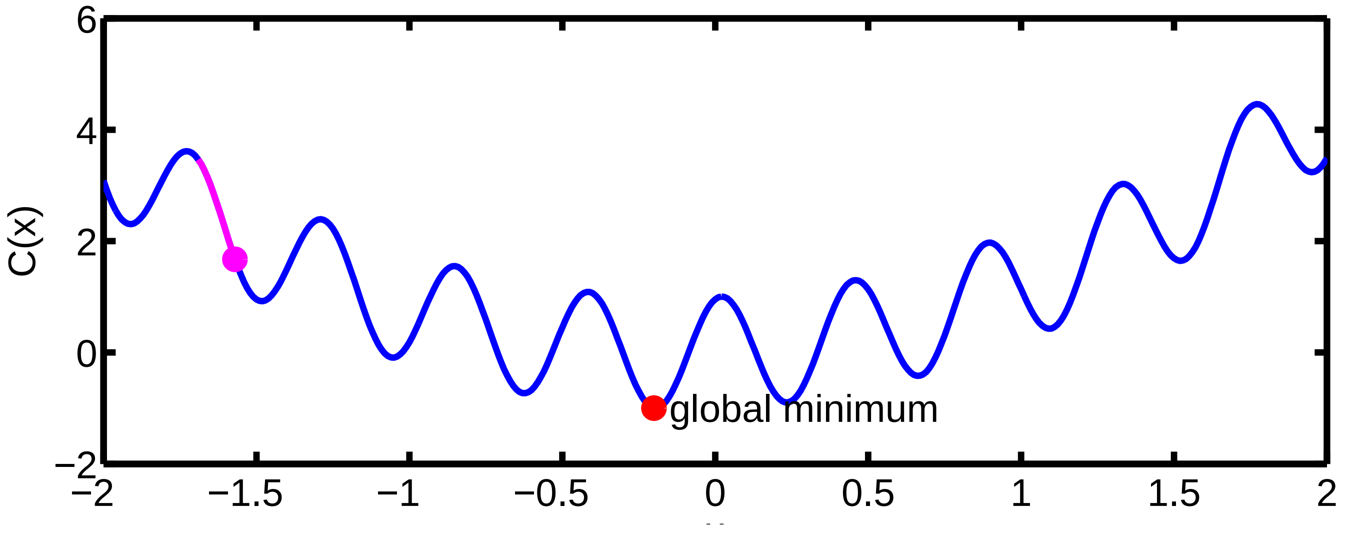
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



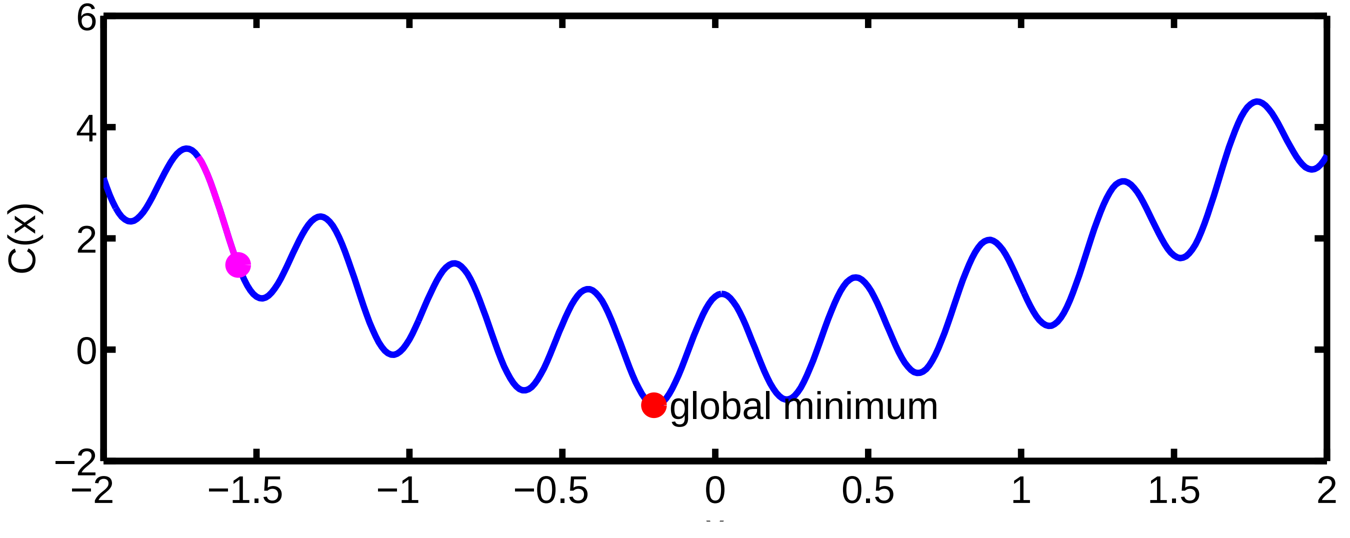
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



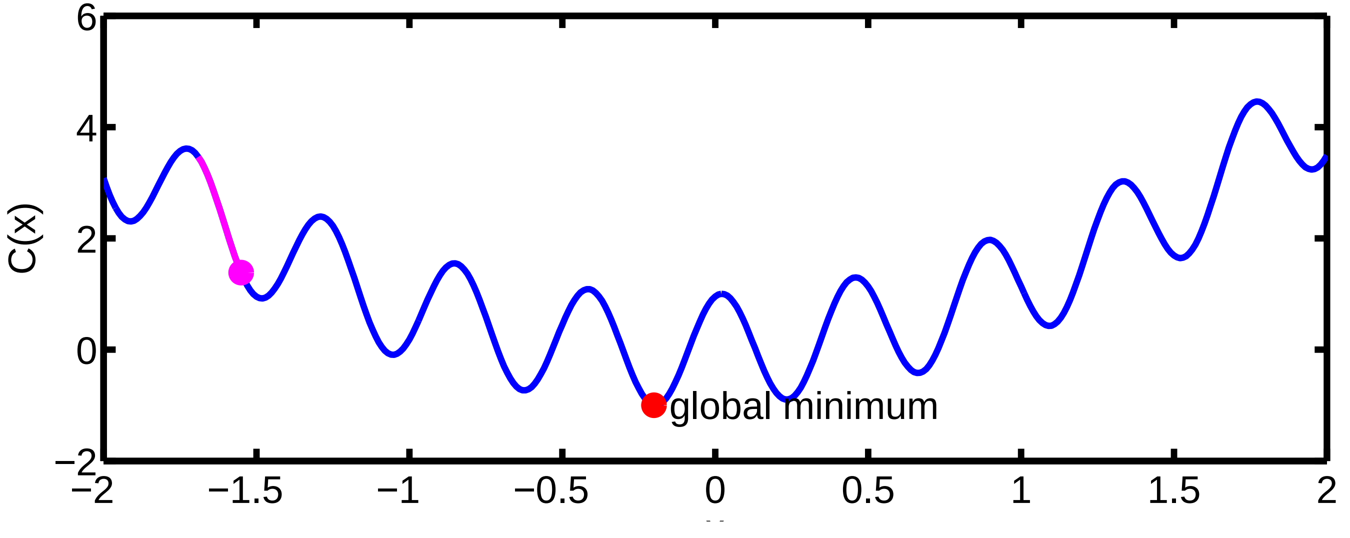
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



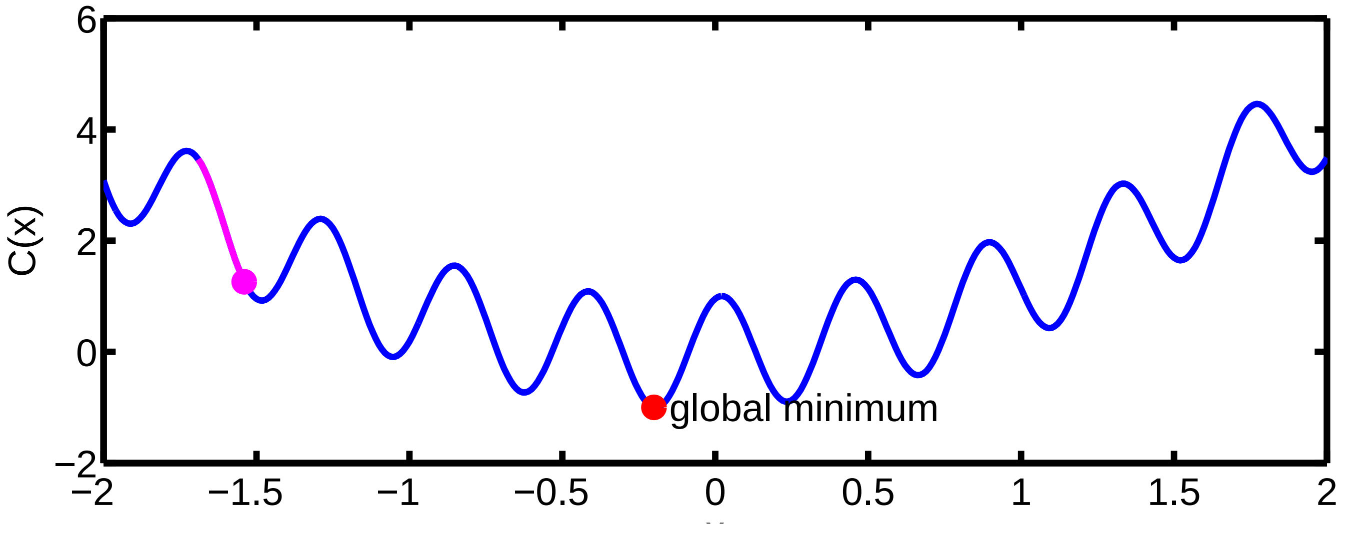
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



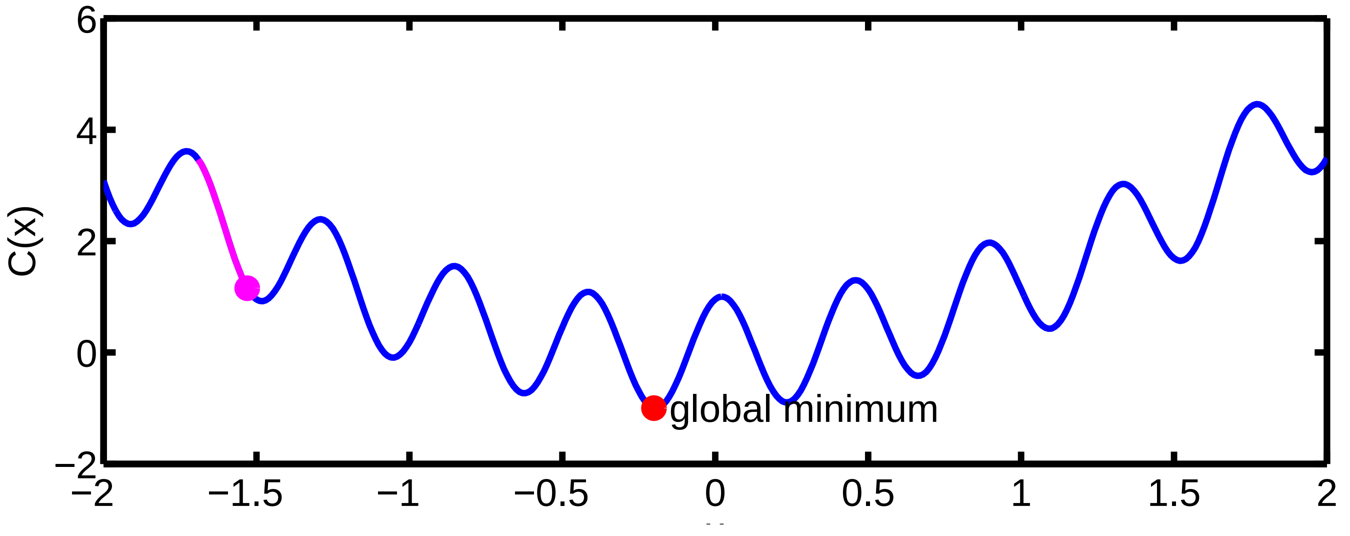
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



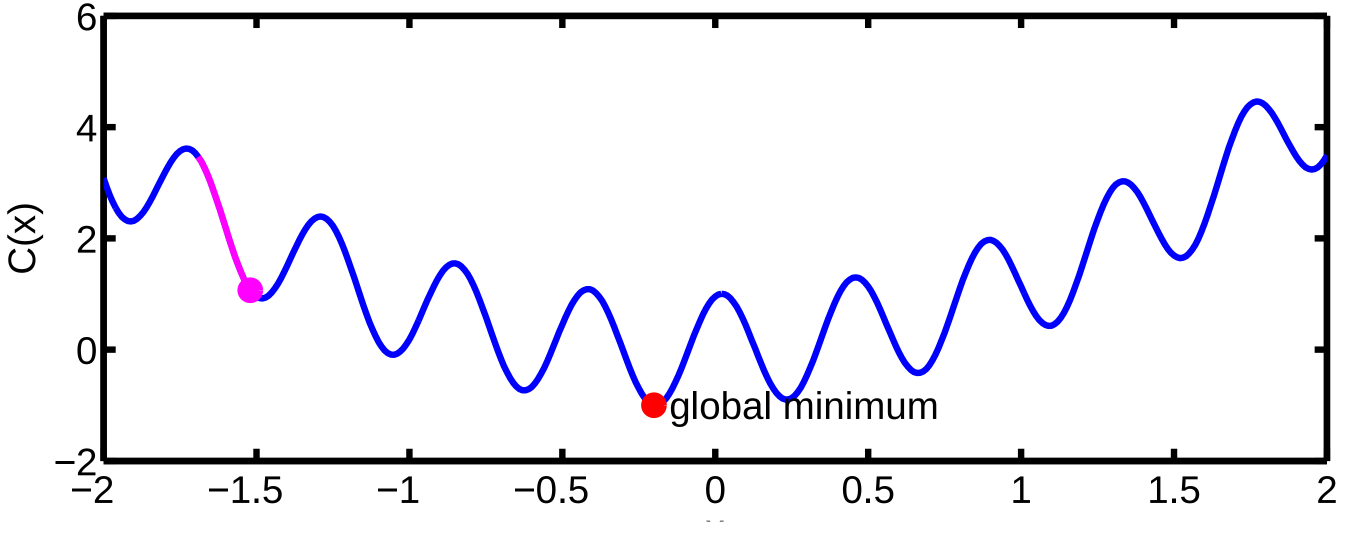
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



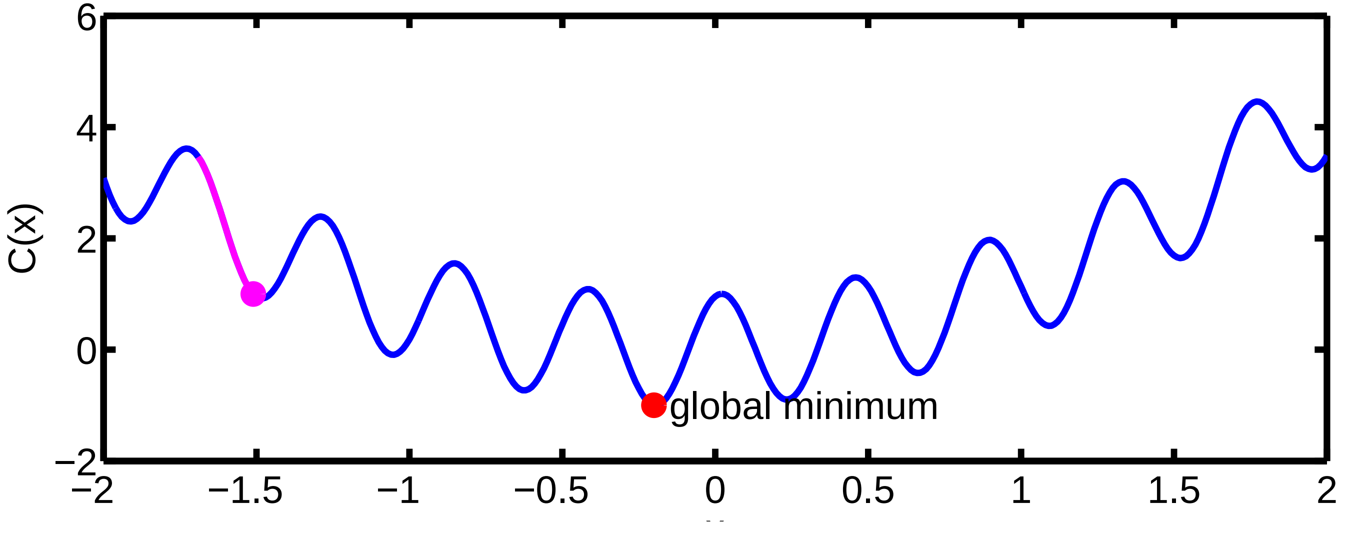
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



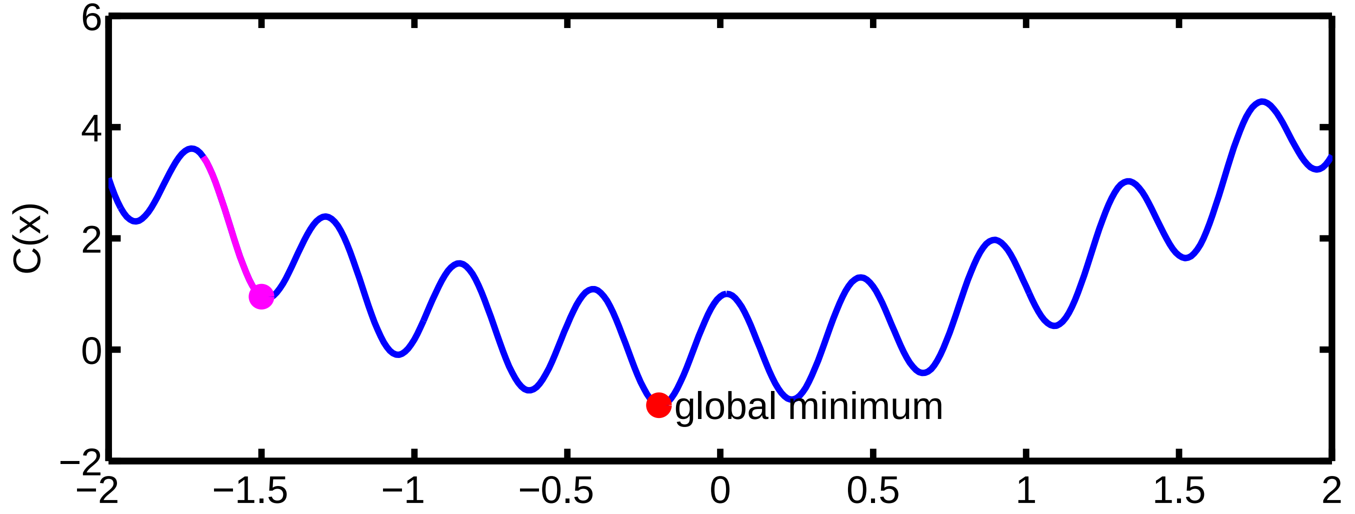
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



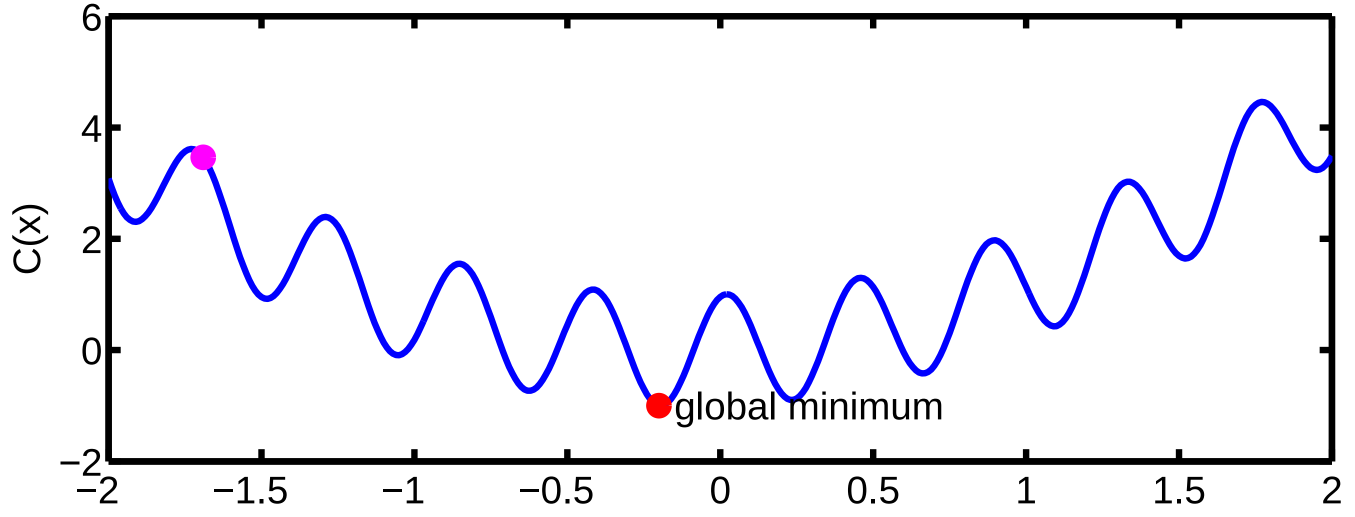
Heuristic methods

- for hard problems we sometimes use heuristics
  - for instance, greedy heuristic
  - try to reduce cost at each step
  - can get stuck in a local minimum



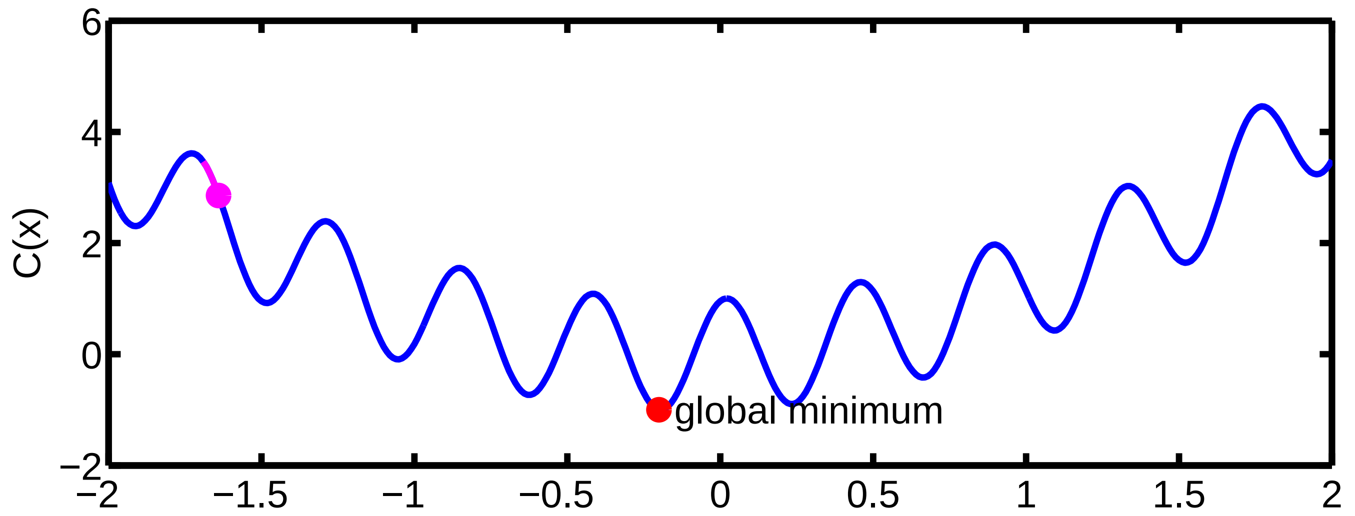
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



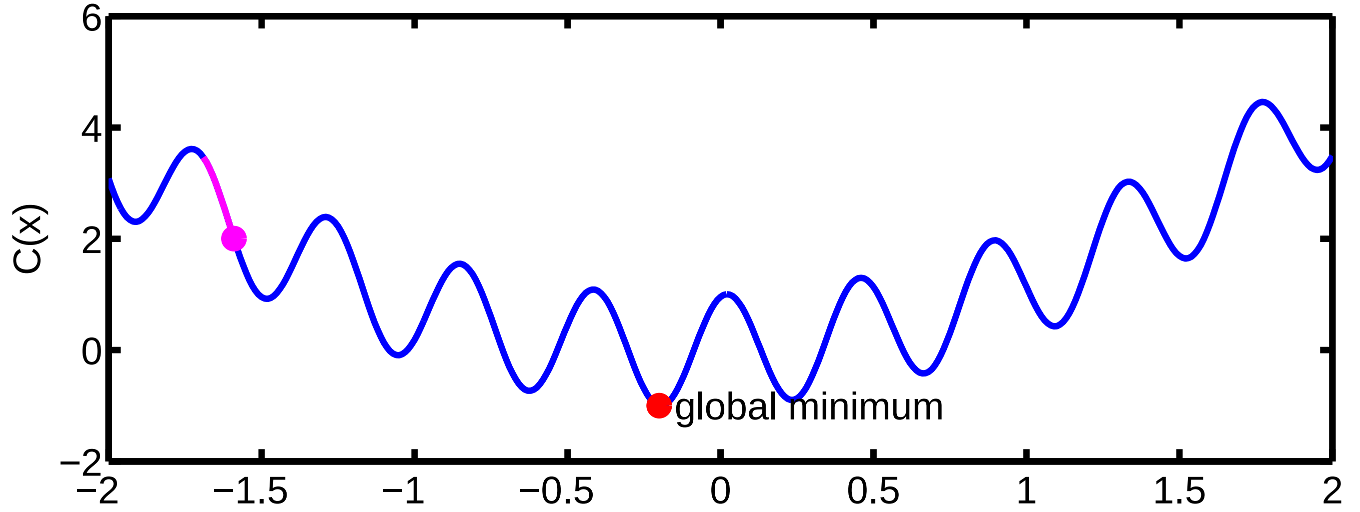
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



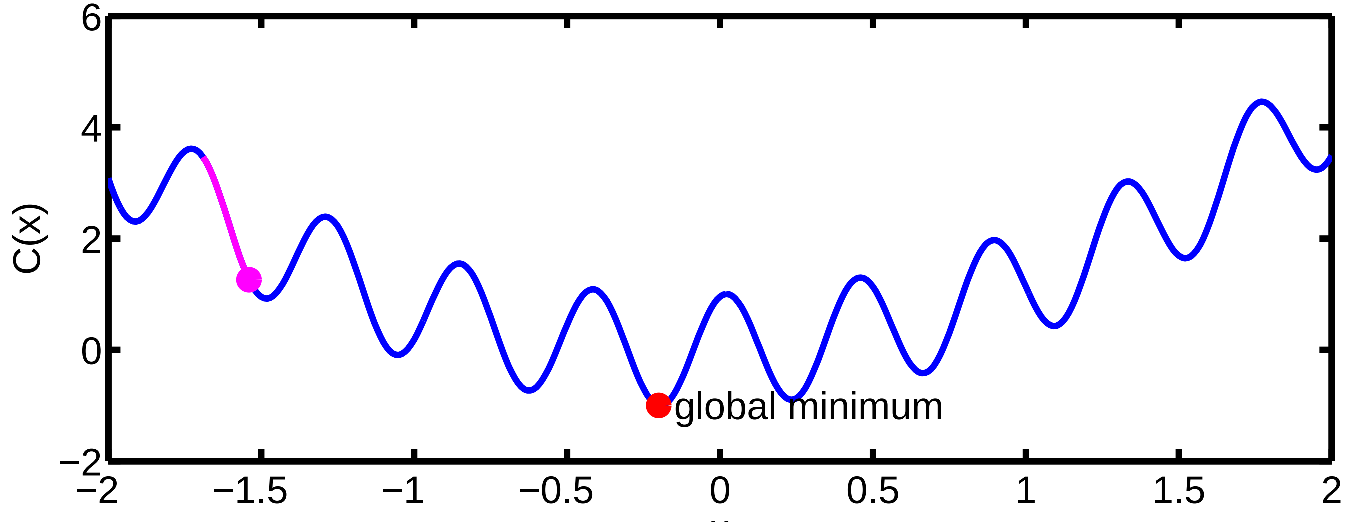
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



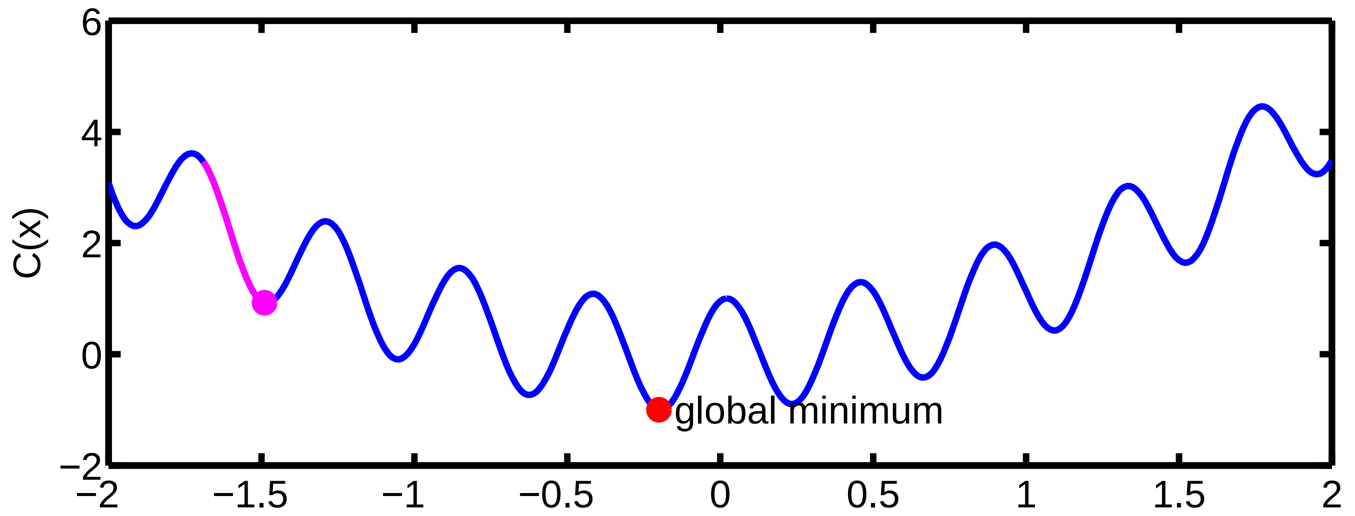
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



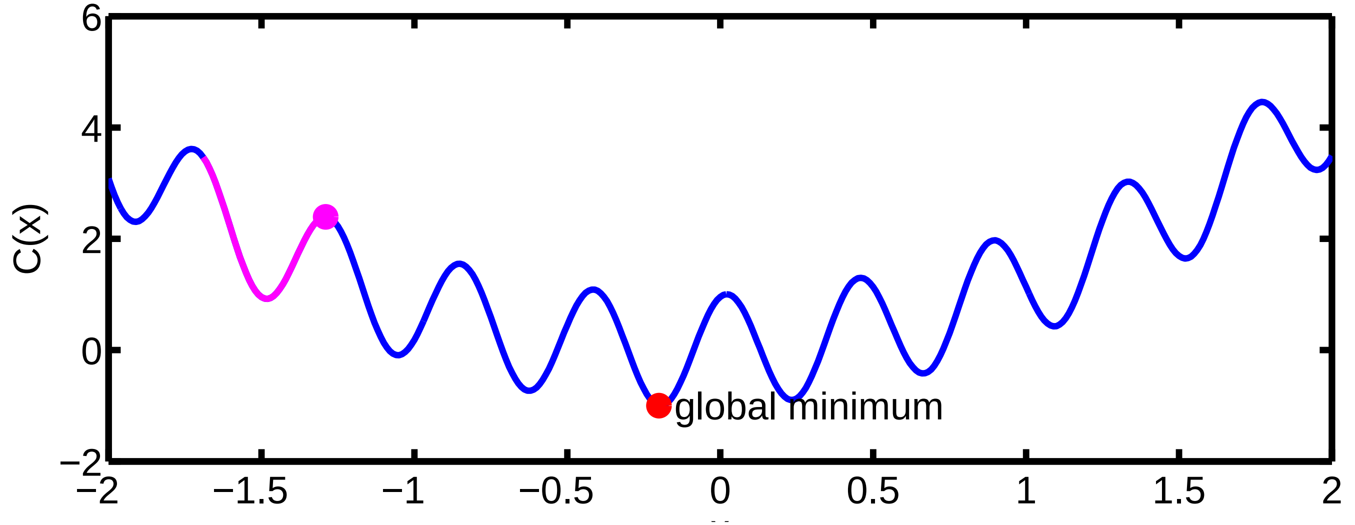
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x + \Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x + \Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



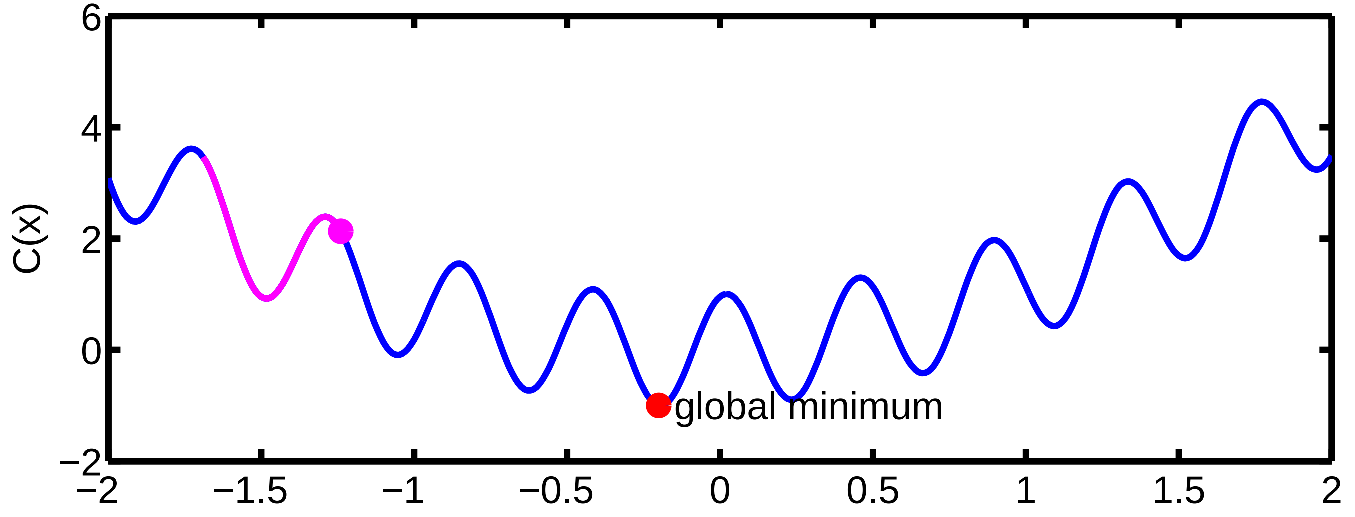
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



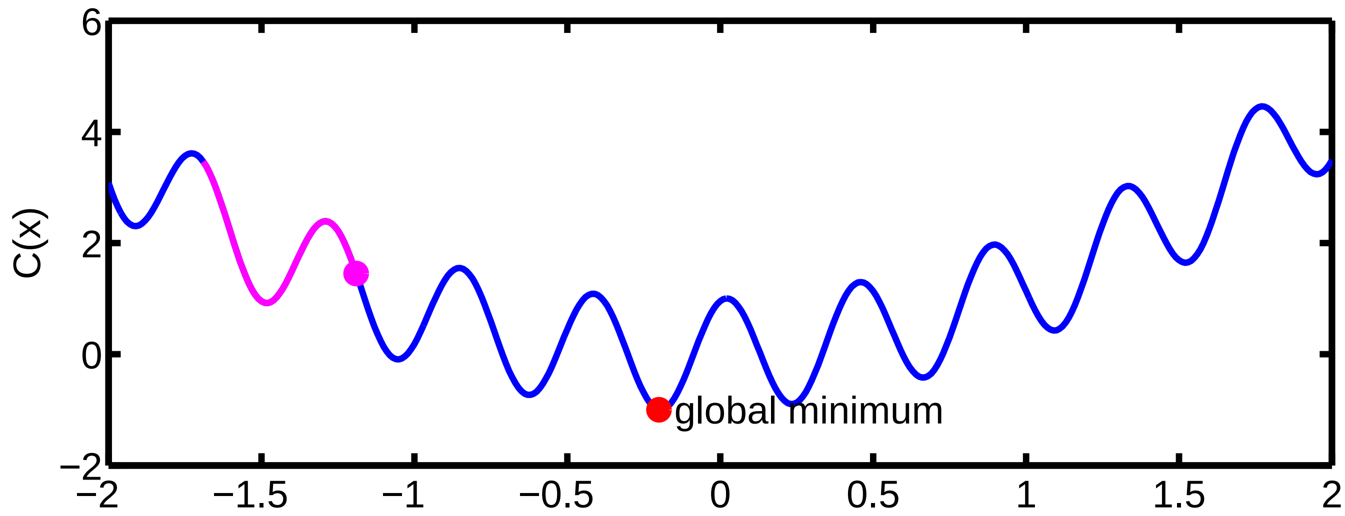
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



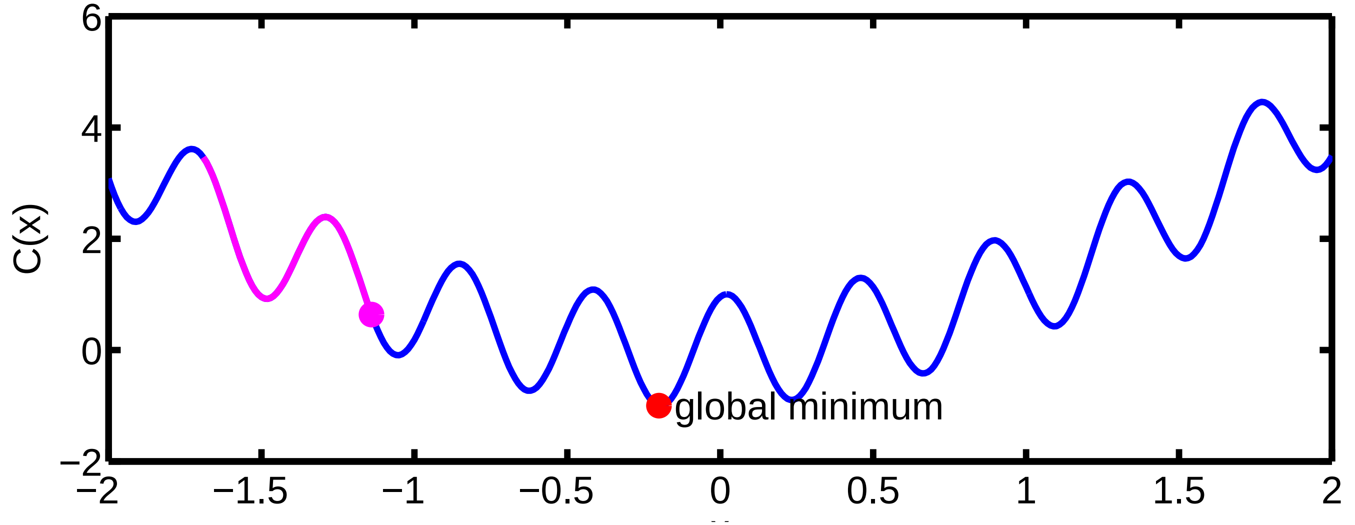
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



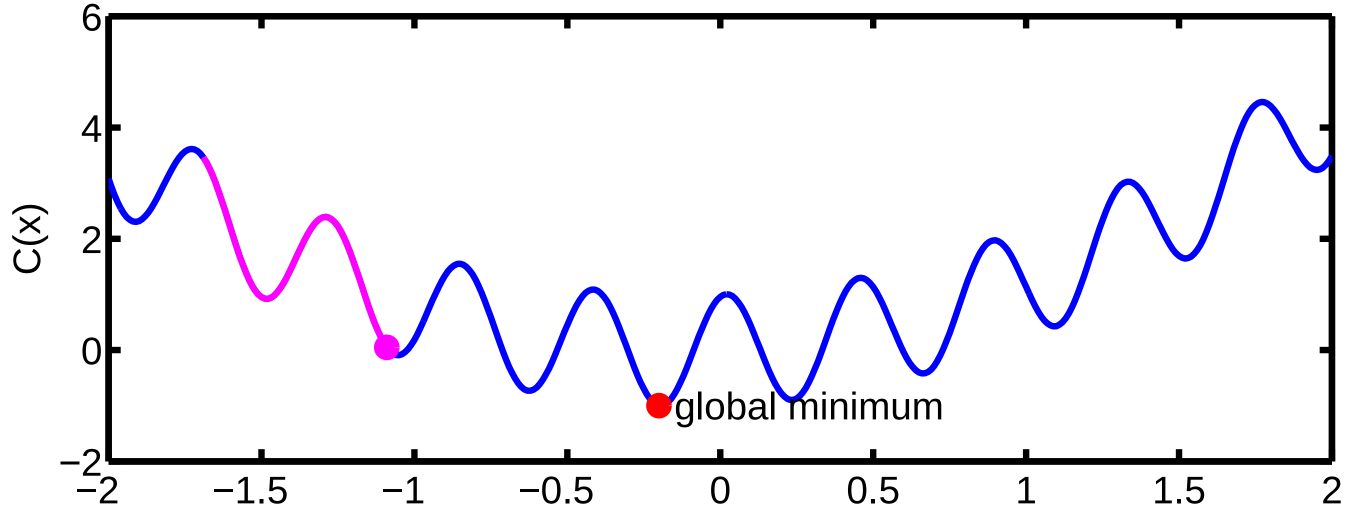
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



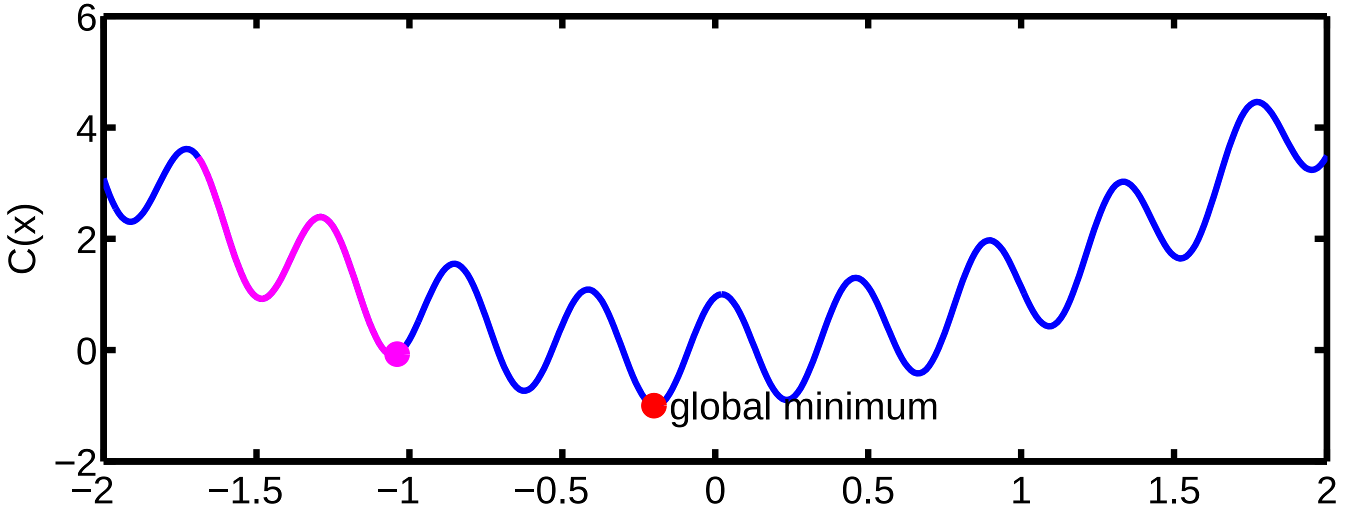
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x + \Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x + \Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



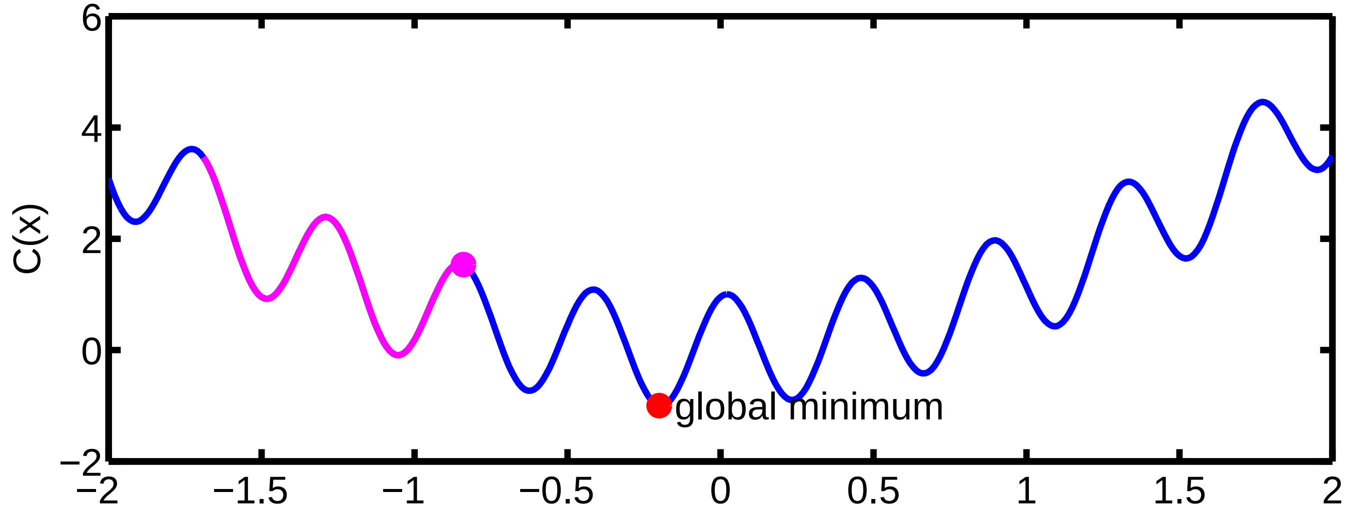
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



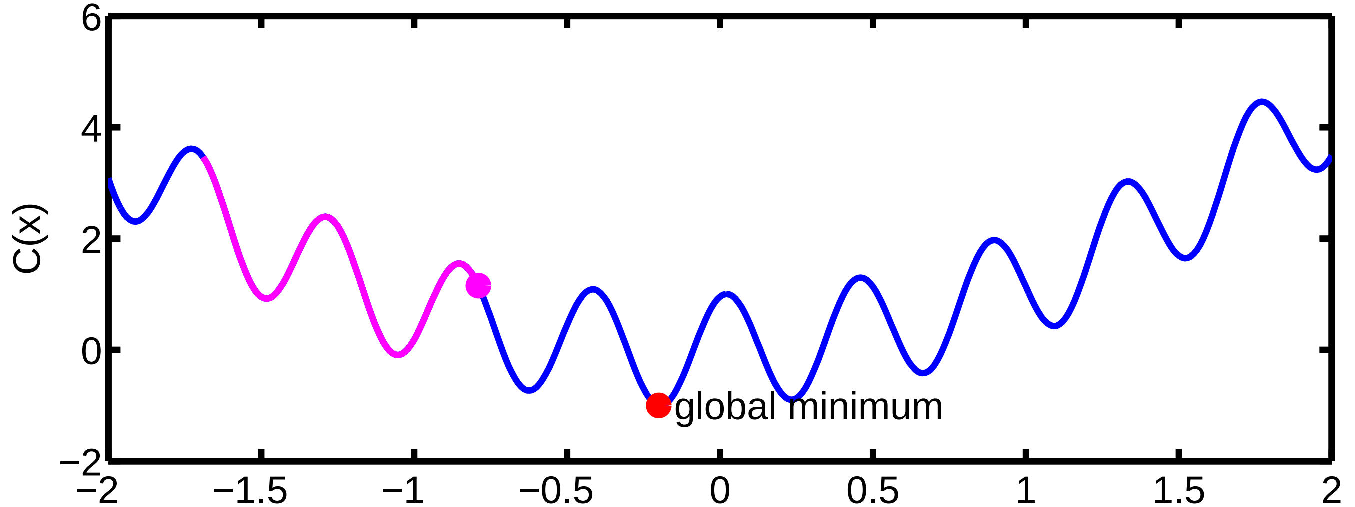
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



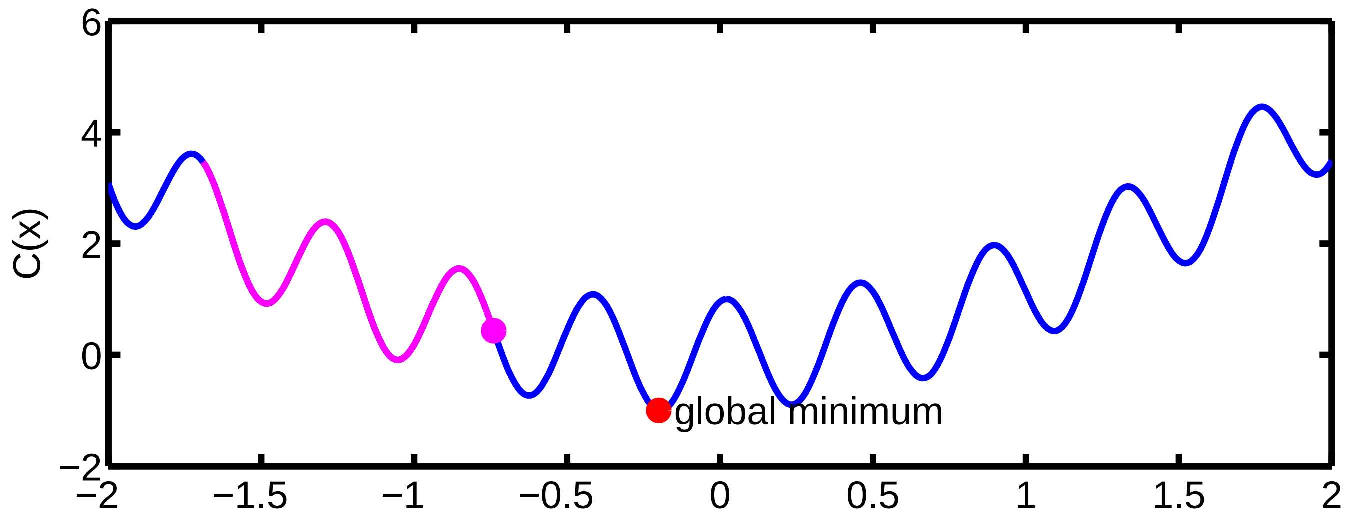
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



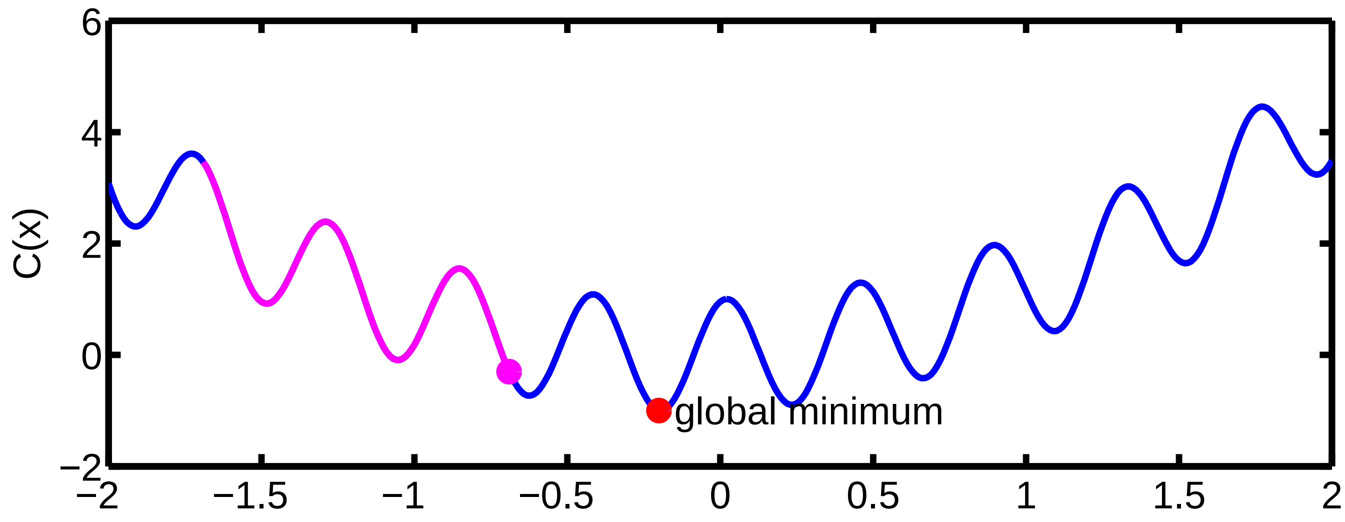
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



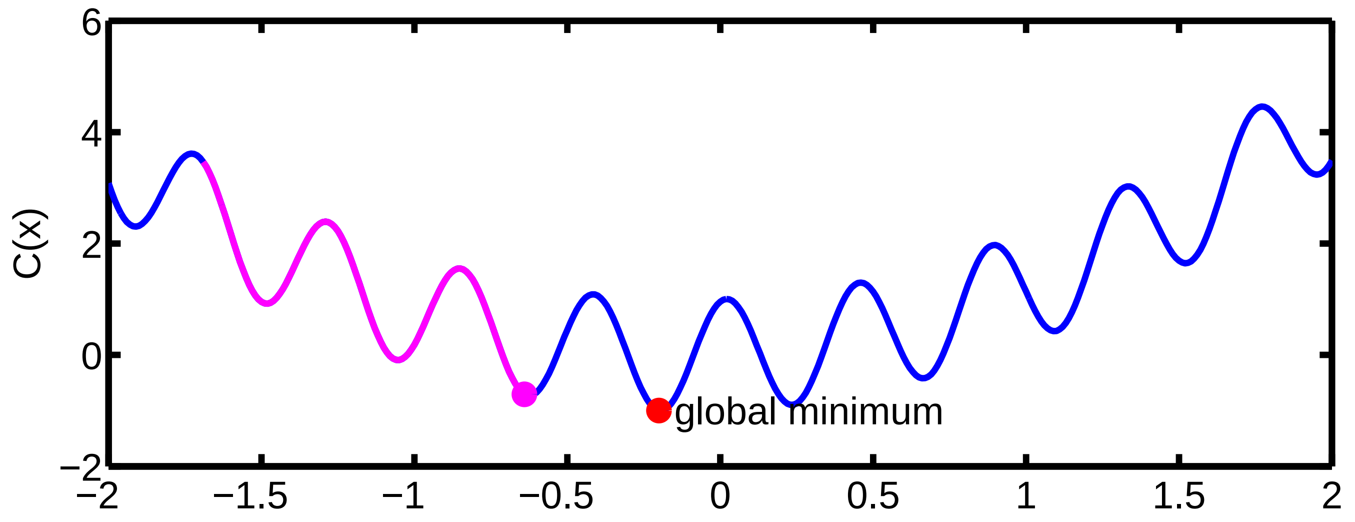
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



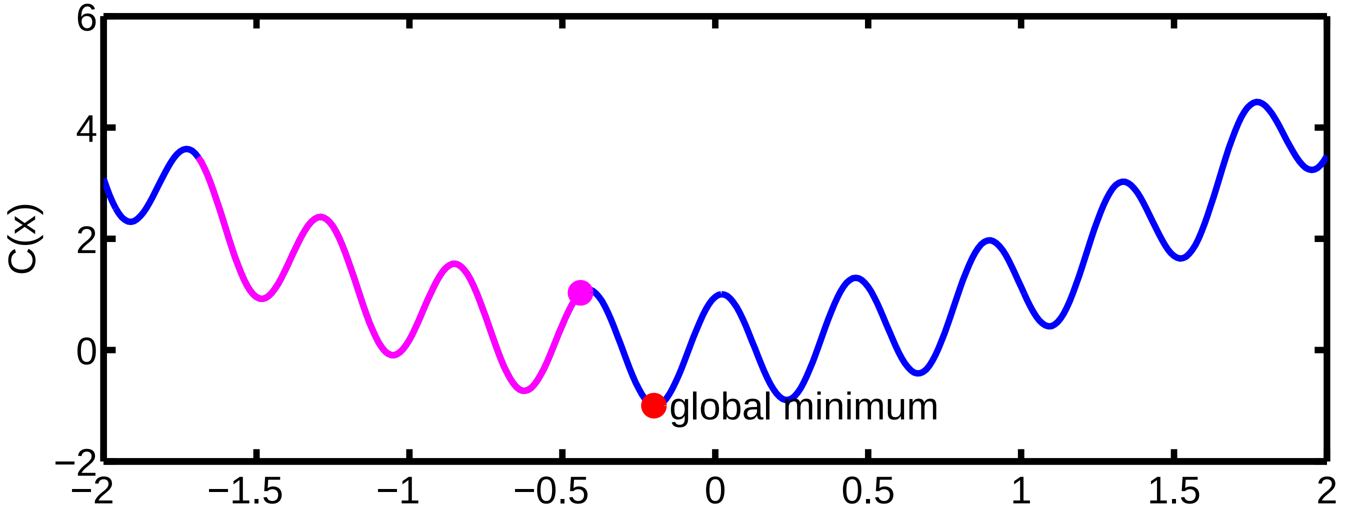
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



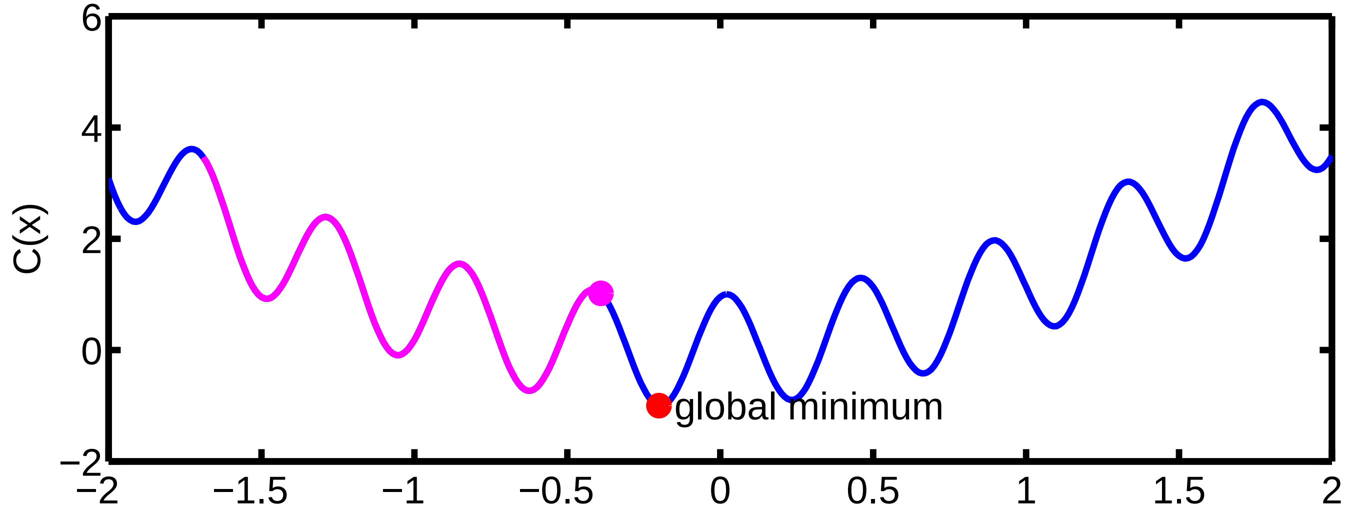
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



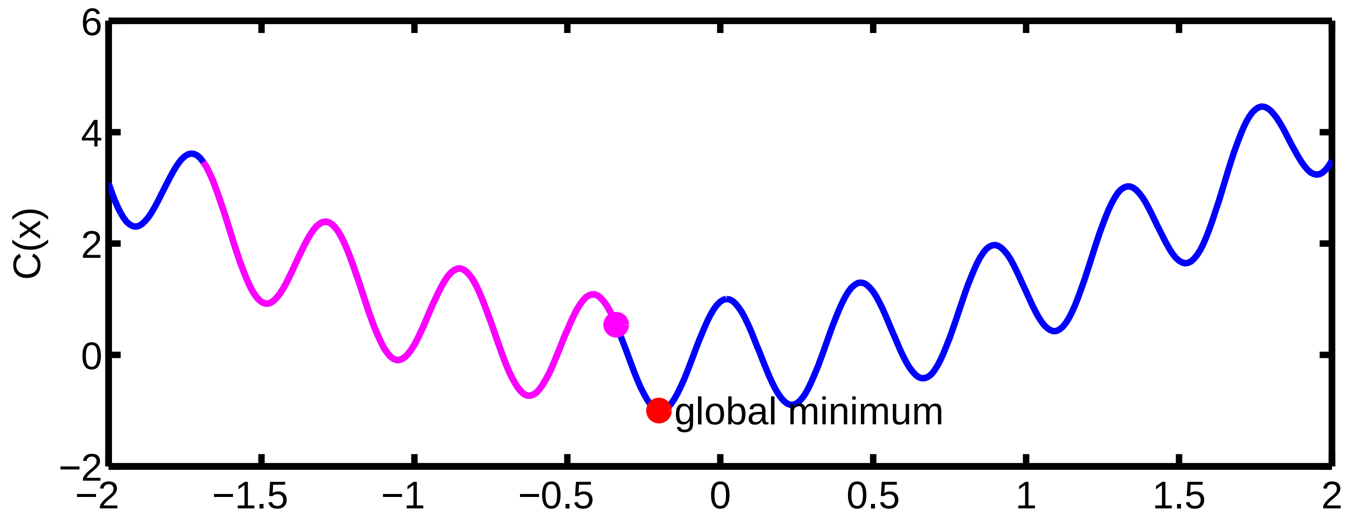
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



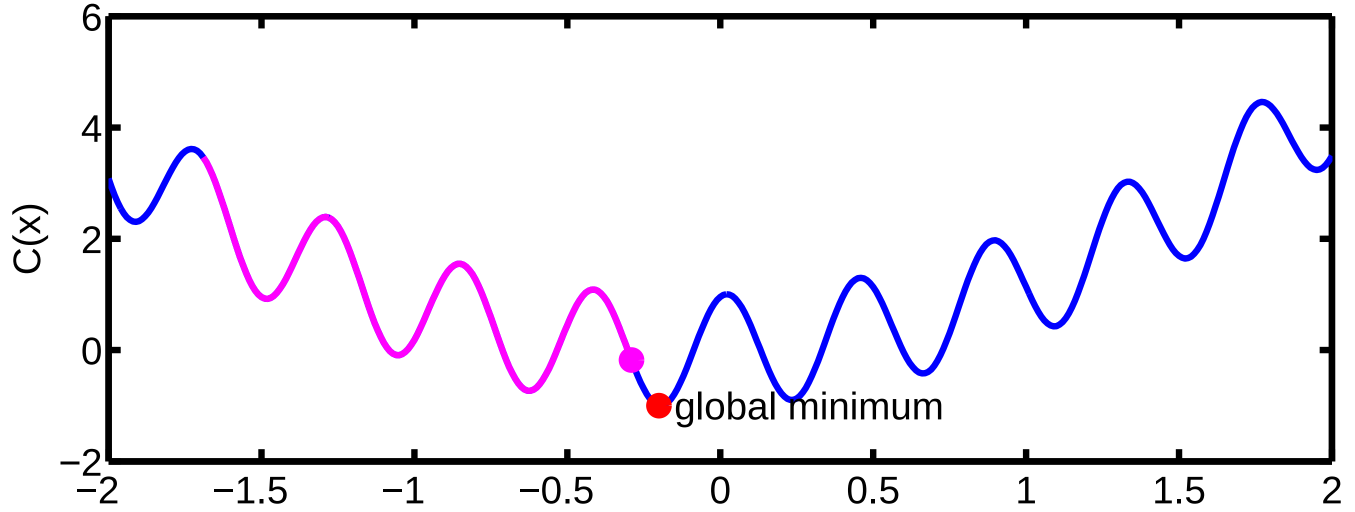
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



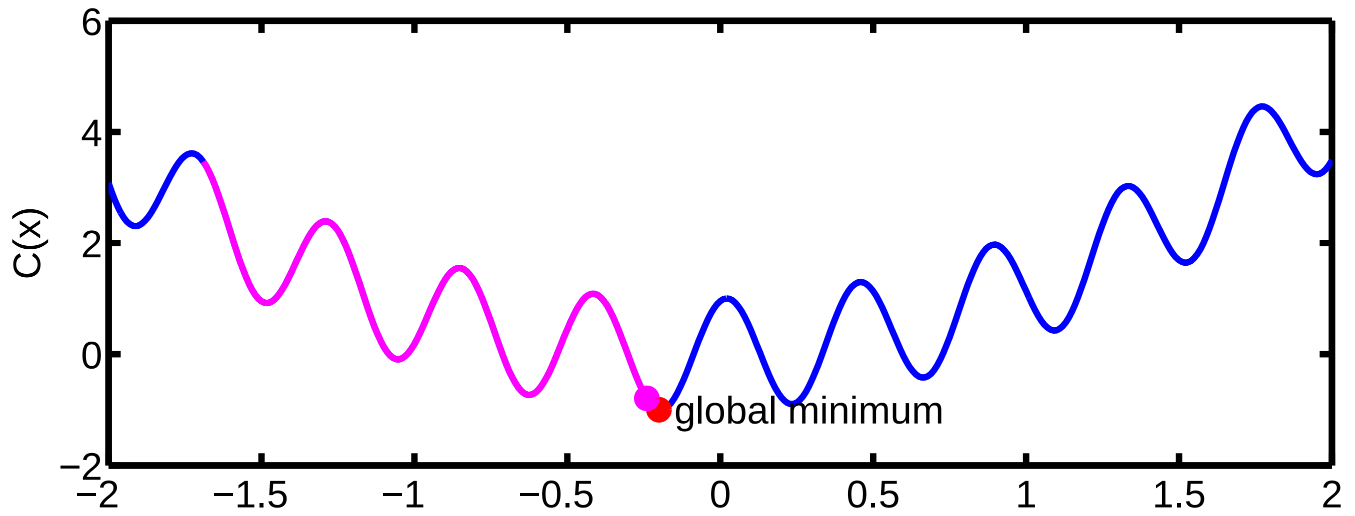
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



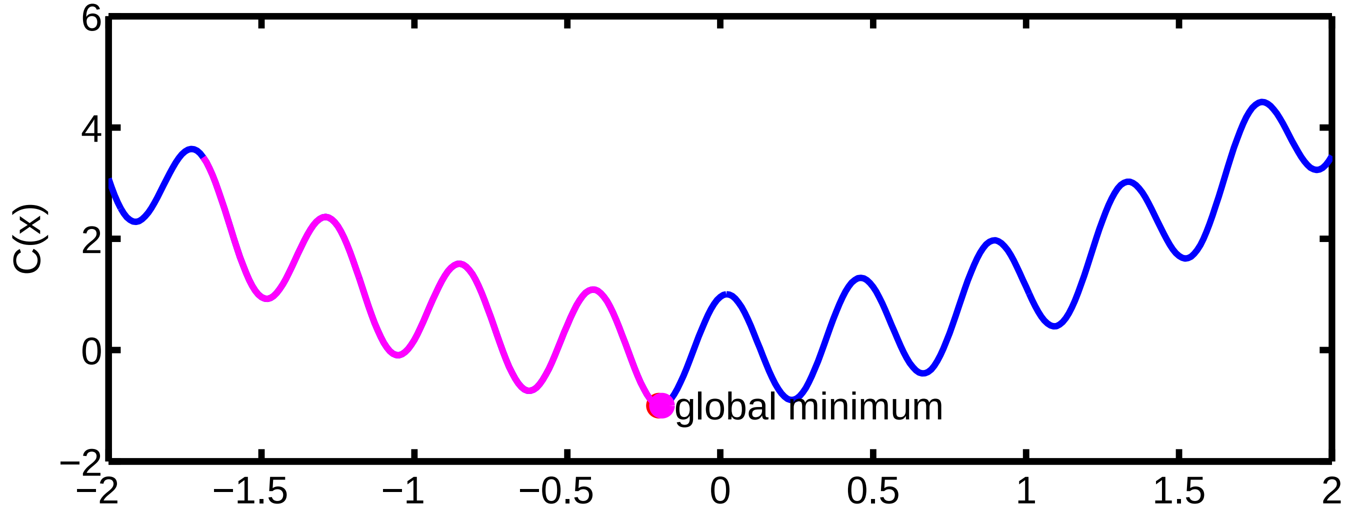
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x + \Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x + \Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



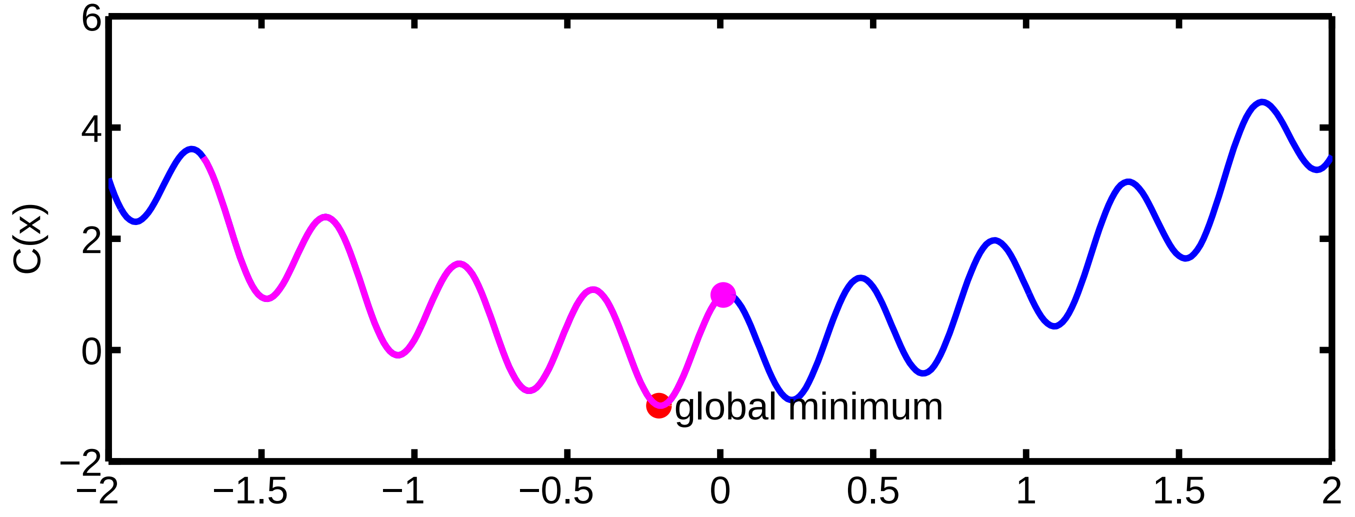
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



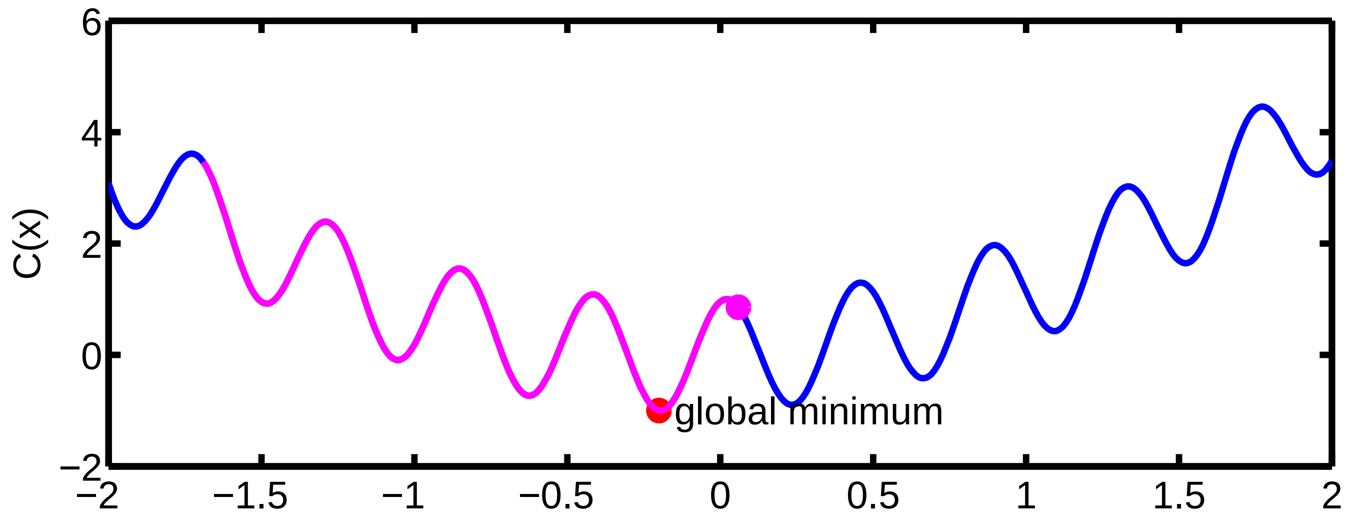
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



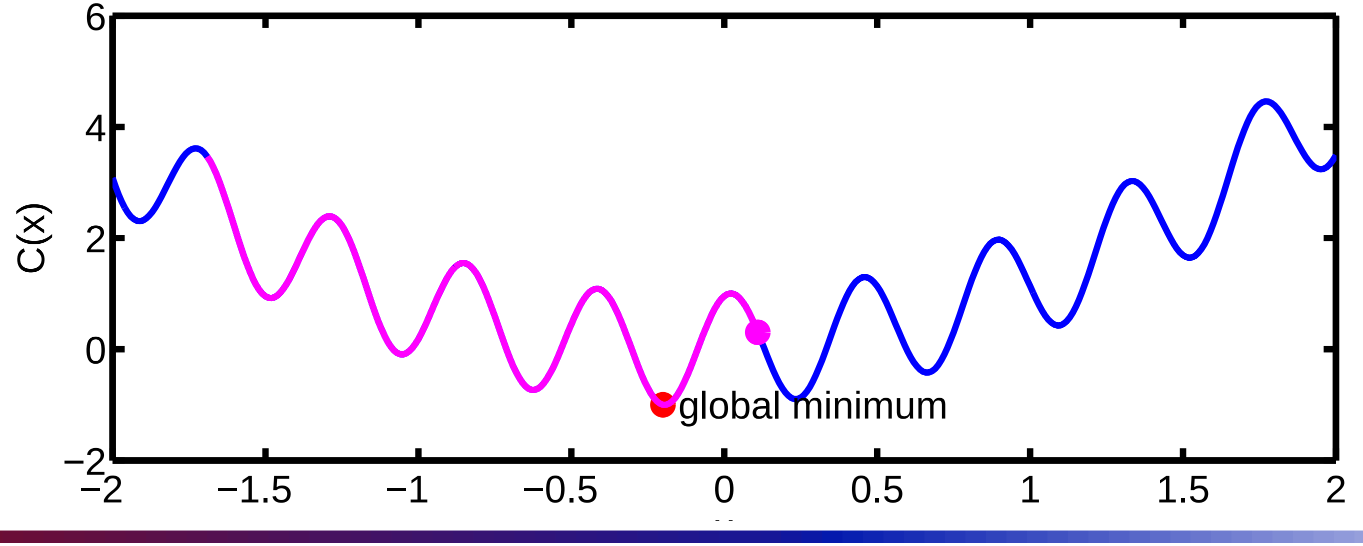
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



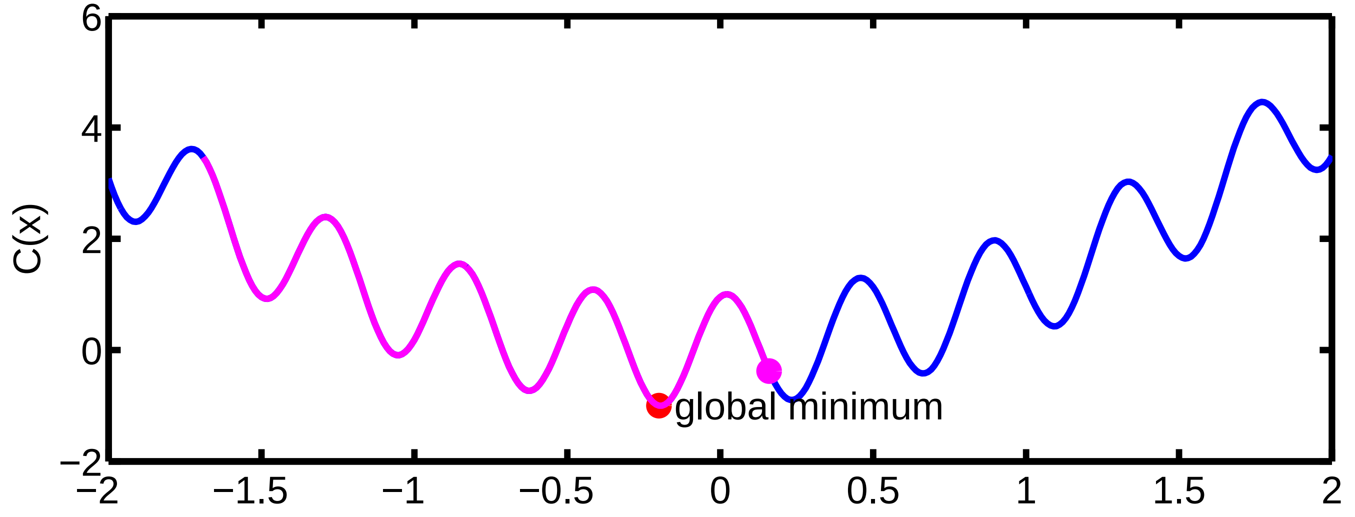
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



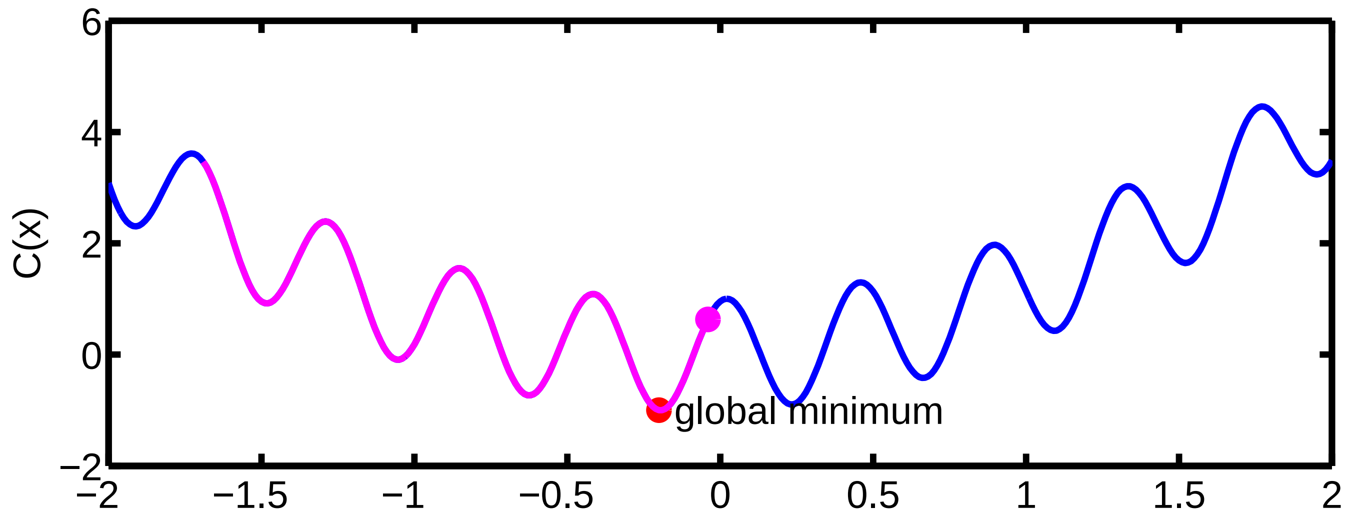
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



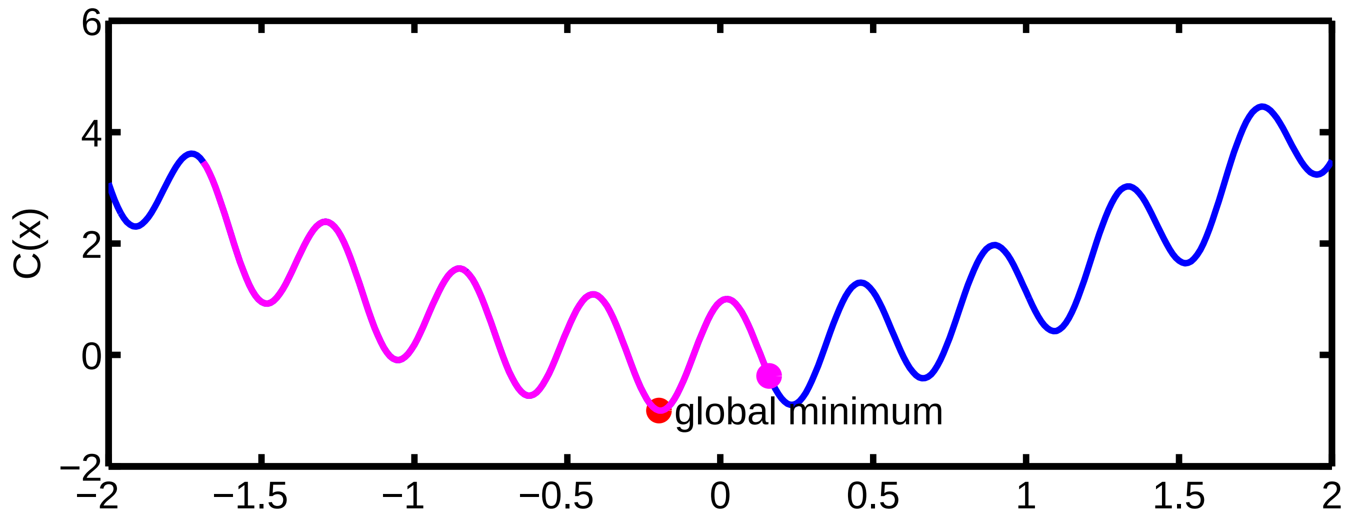
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



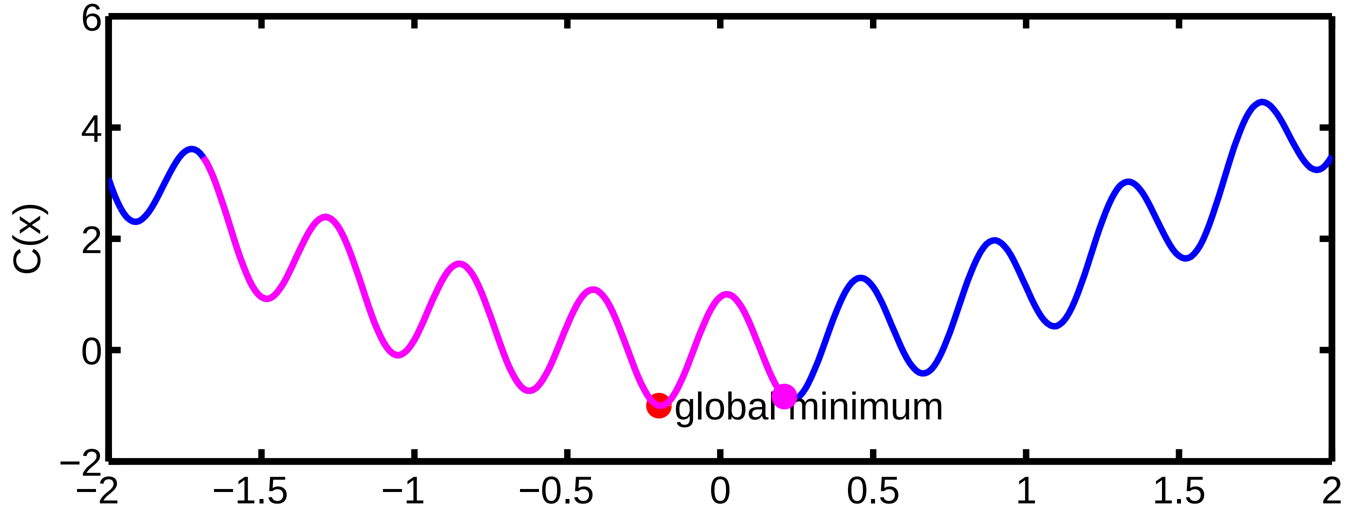
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



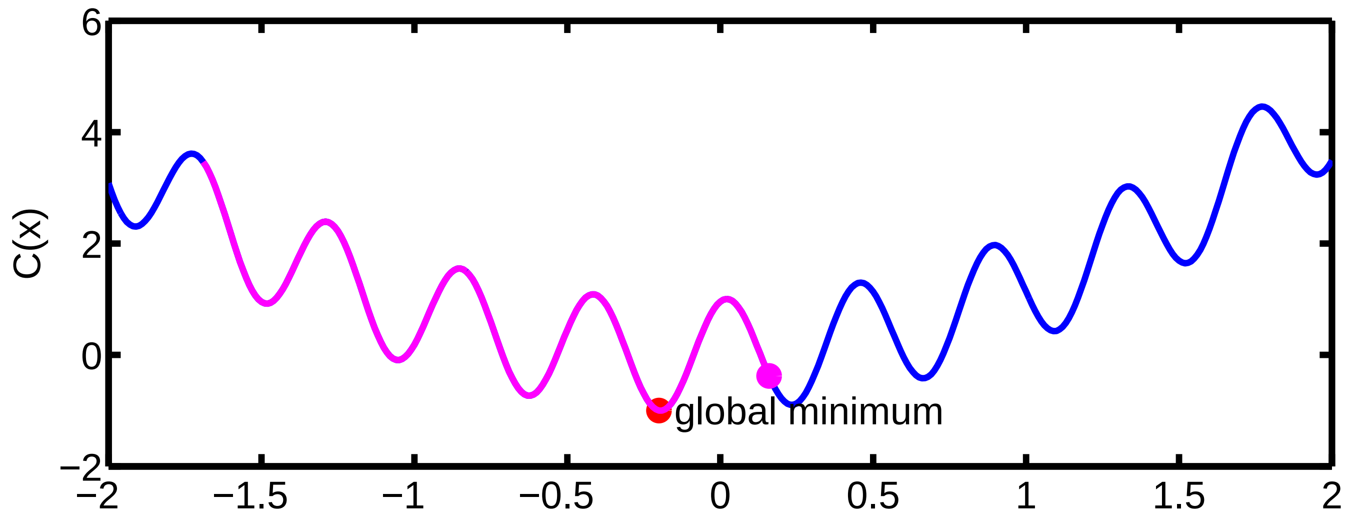
Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



Random search methods

- allow steps that make cost worse
  - normally we always take $C(x+\Delta x) \leq C(x)$
  - random methods sometimes take step Δx such that $C(x+\Delta x) > C(x)$
- examples
  - **Simulated annealing** today [1, 2]
  - **Genetic algorithms** next lecture



Randomized algorithm

- "divide and conquer" is another approach
  - problem needs to separate into subproblems
  - requires detailed insight into the problem
- greedy method gets stuck in a local minimum
  - clever heuristic might be better, but too complex, or we don't know enough about the particulars of the system
  - allow some "random moves", away from improved cost
  - these might just get us out of the local minimum
  - we might just scale that next hill, and go into the deeper valley

Notation

- $\mathbf{x}_i$ is the solution after i iterations
- $C(\mathbf{x})$ is the cost function
- $\mathbf{x}_{i+1} - \mathbf{x}_i = \Delta\mathbf{x}$
- so the cost after $i+1$ steps is given by $C(\mathbf{x}_i + \Delta\mathbf{x})$
- the change in cost is $\Delta C = C(\mathbf{x}_i + \Delta\mathbf{x}) - C(\mathbf{x}_i)$
- T will refer to "temperature"

Simulated annealing

Based on an analogy:

- in Statistical Mechanics and Chemistry **Annealing** is a process for obtaining low energy states of a solid
  - heat a material until it melts
  - reduce temperature gradually, (the process has to be slow enough when near freezing point)
- Temperature reduction too quick
  - the system will be out of equilibrium
  - flawed crystals in solid (not lowest energy state)
  - analogous to a local minimum
- reduce temperature slowly
  - substance takes structure with **least** potential energy
  - analogous to optimization (we want least cost)

Details of the analogy

A simple overview to explain how the annealing works:

An atom in a heat bath is given a small random displacement, with a resultant change ΔE in energy.	A solution to the optimisation problem is changed slightly to give a neighbouring solution, with a change in the cost function of ΔC=new cost-old cost
If $\Delta E \leq 0$, accept displacement and start again	If $\Delta C \leq 0$, accept new solution and start again.
If $\Delta E > 0$, sometimes accept/ sometimes reject the new displacement on the basis of some probability measure.	If $\Delta C > 0$, sometimes accept/ sometimes reject the new solution on the basis of some probability measure.
Either reiterate at this temp. or drop temp.	Either reiterate at this cost or drop cost.

Simulated annealing applications

This sort of method has proved successful in many applications of Optimisation e.g.

- TSP
- Job Shop Scheduling
- Graph Partitioning
- minimum spanning trees in communications networks
- scheduling of 4th year exams
- etc.

Simulated annealing components

Components

- **description of system:** $\mathbf{x}$ in a form we can work with
- **cost function:** $C(\mathbf{x})$
- **random move generator:** rearrangement of existing configuration, to get a neighbouring one.
- **annealing schedule:** The concept of temperature is included via a control parameter to simulate the temperature changes in the annealing process.
  - give temperatures T
  - length of time at a given temperature
- **acceptance function:** when should we (randomly) accept a new solution, given the change in cost

Acceptance function

A greedy acceptance function looks like

$$\begin{array}{rcll} \Delta C & \leq & 0 & \text{accept} \\ \Delta C & > & 0 & \text{reject} \end{array}$$

We can rewrite this in terms of **probability of acceptance**, $P(\Delta C)$, which in this case would be given by

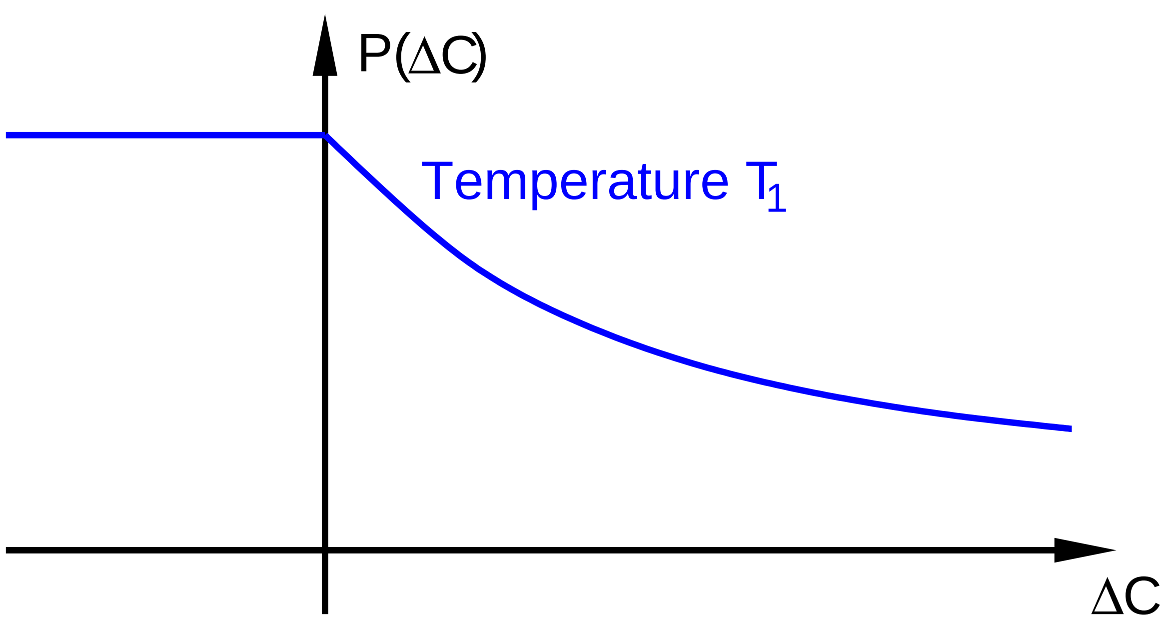
$$P(\Delta C) = \begin{cases} 1, & \Delta C \leq 0 \\ 0, & \Delta C > 0 \end{cases}$$

But we want an acceptance function that will sometimes allow cost-increasing solutions.

Acceptance function

Desirable properties for acceptance function:

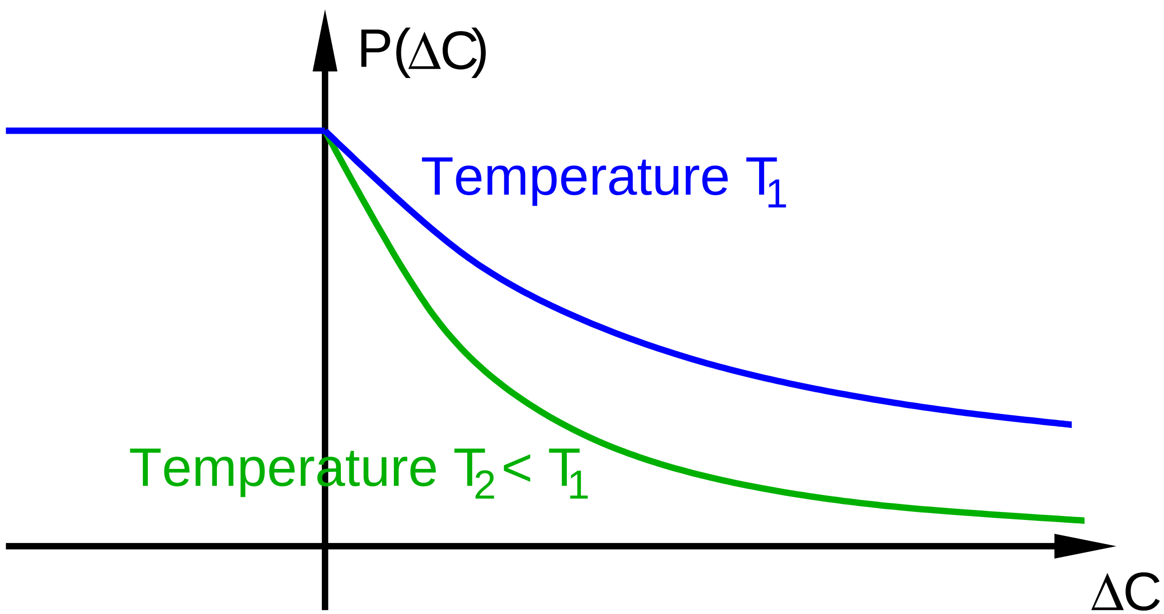
- $P(\Delta C) = 1$ for $\Delta C \leq 0$
- for $\Delta C > 0$
  - $P(\Delta C)$ should decrease as ΔC increase
    - make big increases in cost less likely
  - $P(\Delta C)$ should decrease as T decreases



Acceptance function

Desirable properties for acceptance function:

- $P(\Delta C) = 1$ for $\Delta C \leq 0$
- for $\Delta C > 0$
  - $P(\Delta C)$ should decrease as ΔC increase
    - make big increases in cost less likely
  - $P(\Delta C)$ should decrease as T decreases



Acceptance function

A commonly used acceptance function

- incorporate the Boltzman factor, derived from statistical mechanics

$$\exp\left(\frac{-E(\mathbf{x})}{kT}\right)$$

  which describes the relative likelihood of configurations $\mathbf{x}$ with energies $E(\mathbf{x})$

  - k is Boltzman's constant

- use a new acceptance function

$$P(\Delta C) = \begin{cases} 1, & \Delta C \leq 0 \\ \exp\left(\frac{-\Delta C}{kT}\right), & \Delta C > 0 \end{cases}$$

  In optimization, the temperature is arbitrary, so we may omit the constant k

Acceptance function

Concise way of writing acceptance function

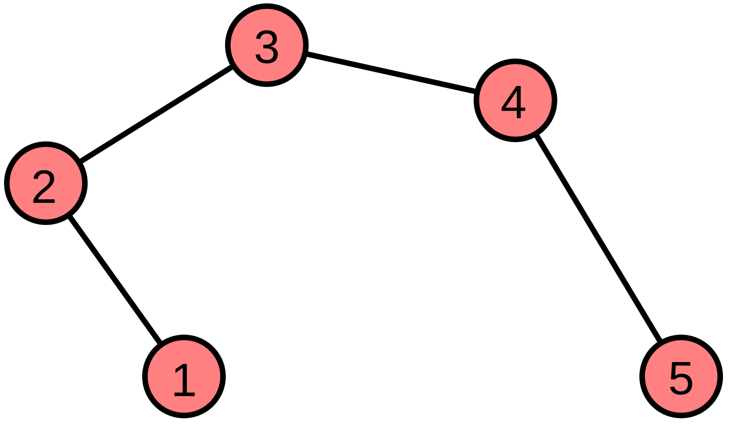
$$P(\Delta C) = \min\left\{1, \exp\left(\frac{-\Delta C}{T}\right)\right\}$$

- incorporate in solution by generating a new neighbouring solution
  - compute the difference in cost ΔC
- generate a uniform random number $p \in [0,1]$
- solution is accepted if $p < P(\Delta C)$

Example of one step

Minimum Spanning Tree Problem $\min C(\mathbf{f}) = \sum_{e \in E} \alpha_e f_e$

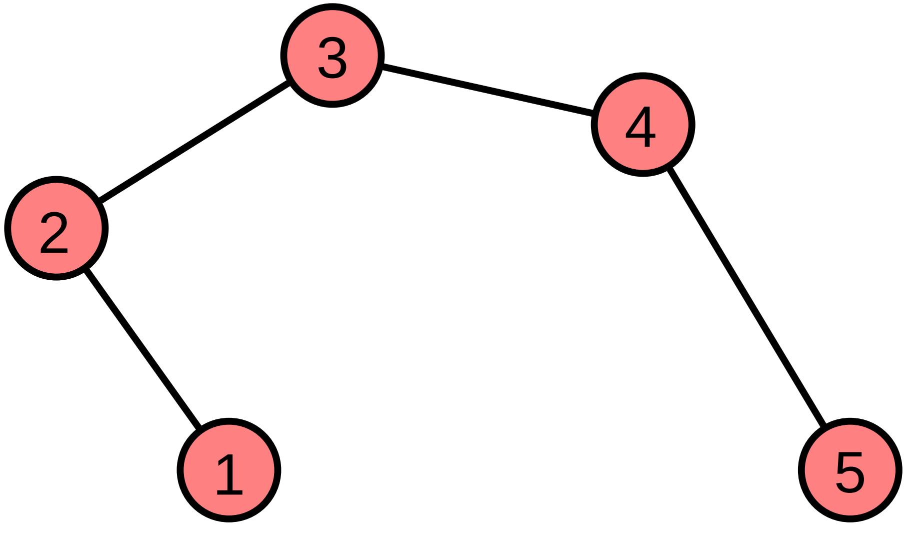
- current solution: a spanning tree
  - choose initially tree where parent of node i is node $i-1$



- generate a neighbouring tree by
  - adding a link e
  - this creates a cycle
  - so remove a link to break the cycle

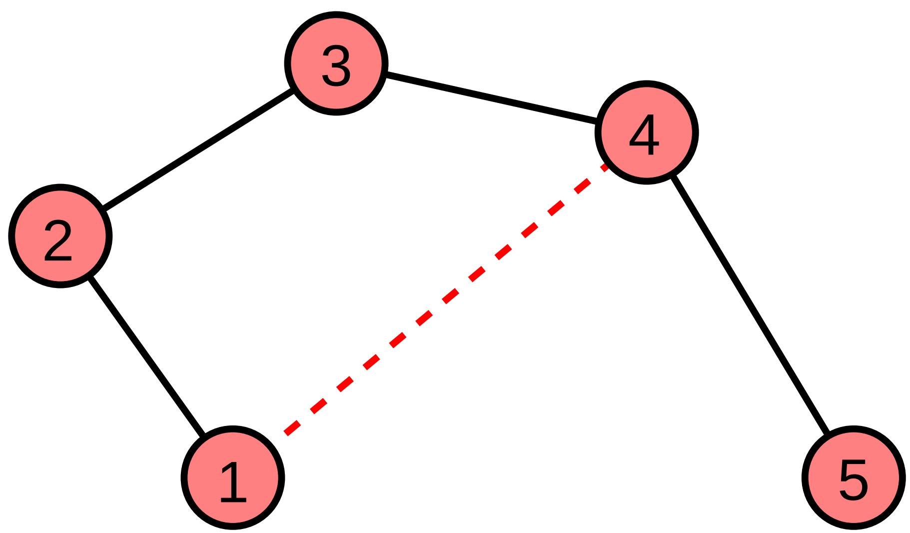
Example of one step

- randomly generate nodes $i, j \in \{1, \ldots, N\}$ and $j \neq i$
  - make sure $e = (i, j)$ is not already in E
  - insert $e = (i, j)$ into E
- now choose a random link e' from the cycle we have created
  - tree won't become disconnected if we remove a link from the cycle



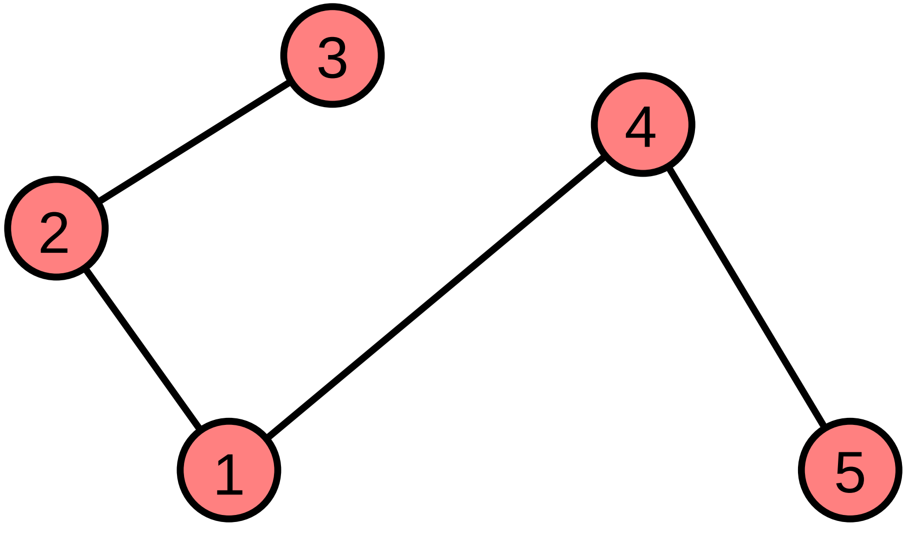
Example of one step

- randomly generate nodes $i, j \in \{1, \ldots, N\}$ and $j \neq i$
  - make sure $e = (i, j)$ is not already in E
  - insert $e = (i, j)$ into E
- now choose a random link e' from the cycle we have created
  - tree won't become disconnected if we remove a link from the cycle



Example of one step

- randomly generate nodes $i, j \in \{1, \ldots, N\}$ and $j \neq i$
  - make sure $e = (i, j)$ is not already in E
  - insert $e = (i, j)$ into E
- now choose a random link e' from the cycle we have created
  - tree won't become disconnected if we remove a link from the cycle



Example of one step

- in example, graph $G(N,E)$, where $N = \{1,2,3,4,5\}$
  - initially $E = \{(1,2),(2,3),(3,4),(4,5)\}$
    - assume it has cost $C = 425$
  - randomly generate two nodes, e.g. 1 and 4
    - $e = (1,4) \notin E$ so we add the link
    - now we have a cycle $1-2-3-4-1$ with 4 links
    - randomly choose one link from the 3 old links of the cycle, e.g. the third link $(3,4)$
    - remove this link from the tree to get E'
  - if $C(E') < C(E)$ accept the new tree, otherwise
    - given current temperature $T = 150$
    - randomly generate $p \sim U(0,1)$
    - say $C(E') = 500$, so $\Delta C = 75$
    - then we would accept E' if $p < e^{-75/150} = 0.607$

Annealing schedule

- in the physical analogy, temperature is reduced slowly over time
  - allows system to stay approximately in equilibrium as the temperature decreases
- we need to do something analogous here
- two methods
  - **homogeneous**: run the above algorithm for a while, and then reduce the temperature, and then repeat.
  - **inhomogeneous**: decrease the temperature at each step.
- also we need a schedule of temperature reductions

Annealing schedule

Two parts of annealing schedule

- initial temperature
  - has to be high enough for "melting"
  - varying proposals as to how hot this should be
    - $P(\Delta C) = 0.5$ for initial neighbours
    - $P(\Delta C) = 0.8$ for initial neighbours
    - could initially test all neighbours to see what temperature is needed
- temperature reductions
  - could give a table of temperature reductions
  - more commonly use geometric decrease

$$T_{i+1} = \alpha T_i$$

where α is usually between $[0.75, 0.95]$

Metropolis algorithm

Idea in Physics/Chemistry [1]
Optimization algorithm first proposed in [2]

- start with random solution $\mathbf{x}$, and temp T_0
- **while** not "frozen"
  - **for** $j = 1, \ldots, J$
    - generate a random neighbouring solution $\mathbf{x} + \Delta\mathbf{x}$
    - find the cost of this solution $C(\mathbf{x} + \Delta\mathbf{x})$, and the change in cost, e.g. $\Delta C = C(\mathbf{x} + \Delta\mathbf{x}) - C(\mathbf{x})$
    - generate a random variable $p \sim U(0,1)$
    - if $p < P(\Delta C) = \min\left\{1, \exp\left(\frac{-\Delta C}{T_i}\right)\right\}$ accept the solution, i.e. $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$
  - $T_{i+1} = \alpha T_i$

TSP Example [2]

Travelling Salesman Problem (TSP) from Lecture 12

- state is the z_e (do we use link e)
- moves to neighbours by
  - reversing the direction in which a part of the tour is traversed [3]
  - this move preserves constraints
  - other possibilities exist
- initial $T = O(N^{1/2})$, where moves flow around freely
- in 1983, sim.annealing could (approximately) solve a 6000 node problem
  - best exact solution for 318 nodes

TSP Example

- the above uses a clever move to make sure constraints remain satisfied by a neighbour
- what if we don't know a "clever move"
  - transform the problem to an unconstrained one
  - construct an augmented objective function incorporating any violated constraints as large penalty functions
    - e.g. minimize cost $C(\mathbf{x})$ subject to $\mathbf{x} \geq 0$
    - transform to
      $$\min \left[C(\mathbf{x}) + 10^6 \times I(\mathbf{x} < 0) \right]$$
      where $I(\cdot)$ is an indicator function
  - solutions which violate the constraints will have very high cost

Applet Example

Some nice examples from the web.

http://appsrv.cse.cuhk.edu.hk/~csc6200/y99/applet/SA/annealing.html

http://www.math.uu.nl/people/beukers/anneal/anneal.html

Algorithm issues

- initialization
  - start with a random solution
  - start with a “good” solution, from a heuristic
    - might be faster
    - might also get stuck in a local minima
      - if the temperature doesn't start hot enough
      - but if the temperature is hot enough, why bother?
- if we start with $T_0 = 0$ we get a greedy algorithm

Algorithm issues

- homogeneous approach
  - how many times should we run the inner-loop before changing the temperature
  - long enough to explore the regions of search space that should be reasonably populated
  - actually might need a bit of trial and error to get a number
    - can be problem dependent
    - large problems have a larger solution space
- termination
  - when $T = 0$ things are "frozen" in place
  - or when nothing changes for several outer-loop iterations

Final

Many much more sophisticated modifications of the approach in the literature, e.g. [4]

References

[1] N.Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” J. Chem. Phys., vol. 21, no. 6, pp. 1087–1092, 1953.

[2] S. Kirkpatrick, C. D. Gelatt Jr., and M. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, pp. 671–680, 1983.

[3] S.Lin and B.W.Kernighan, “,” Oper.Res., vol. 21, 1973.

[4] L. Wang, H. Zhang, and X. Zheng, “Inter-domain routing based on simulated annealing algorithm in optical mesh networks,” Opt. Express, vol. 12, pp. 3095–3107, 2004. http://www.opticsexpress.org/abstract.cfm?URI=OPEX-12-14-3095.