UNIVERSITÄT PASSAU

Search-Based Software Engineering

Local Search Algorithms

Gordon Fraser
Lehrstuhl für Software Engineering II

Local Search Algorithms

- Common algorithmic principle:

  1. For a given problem we define a search space S and an objective function $f: S \rightarrow R$

  2. Neighbourhood $V(x)$ for $x \in S$: Set of solutions y that one can reach from x in one step.

     - Usually defined implicitly by a set of transformations T_i that generate y starting from x: if y in $V(x)$ then there is an i for which $y = T_i(x)$. (Transformations = "moves")

  3. Exploration operator U: Applied on a current solution x_0 generates the next point to explore in the search trajectory. Operator U makes use of the fitness values in the neighbourhood to generate the next solution.

  4. Exploration process $x_0 \rightarrow x_1 \in V(x) \rightarrow x_2 \in V(x_1) \rightarrow \ldots$ continues until a stopping criterion is met.

- The result is either the last x_n in the trajectory, or the x_i found along the trajectory that produces the best fitness value.

- Efficiency depends on the choice of $V(x)$, coding of solutions, choice of U, etc.

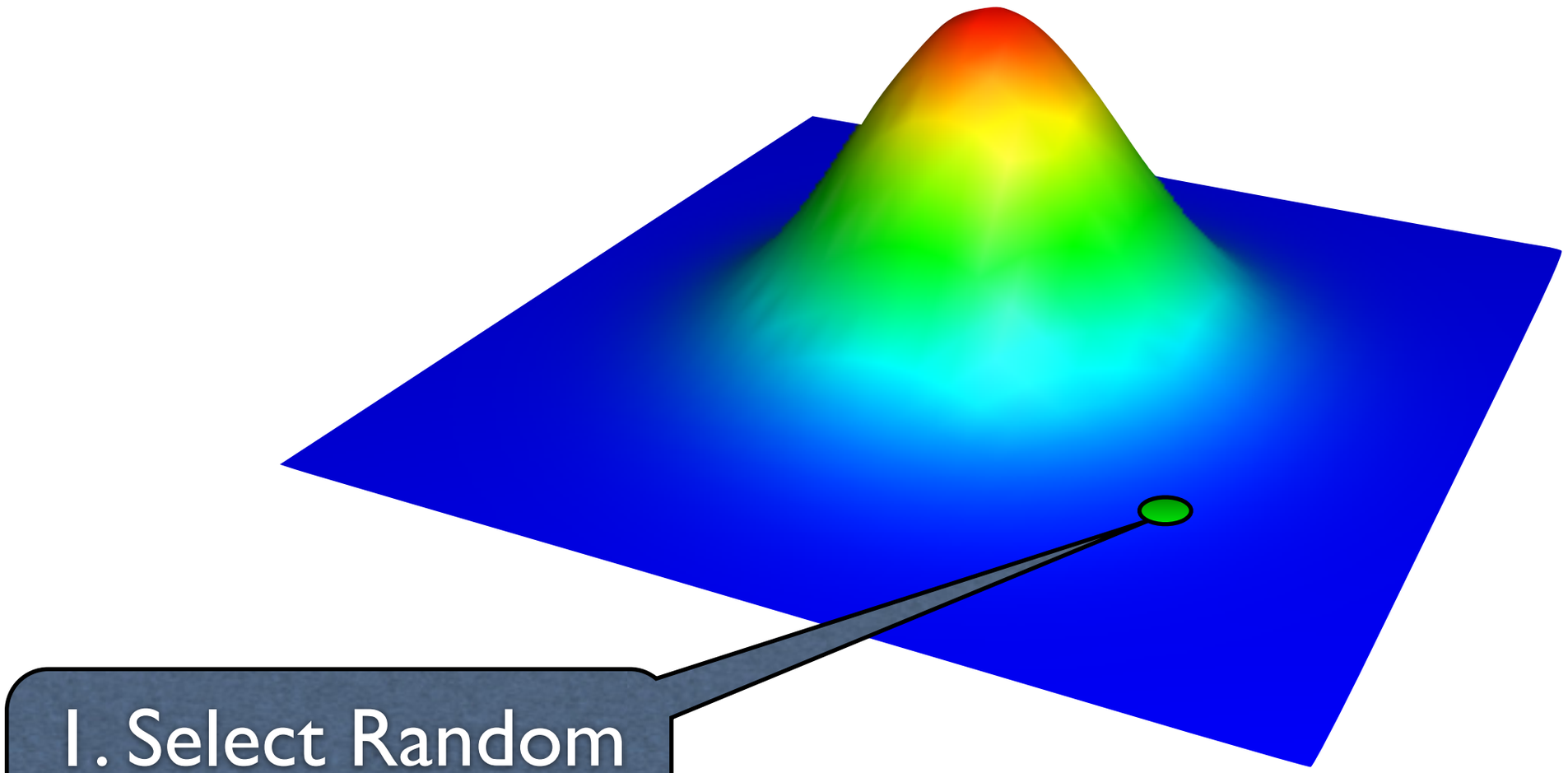
Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

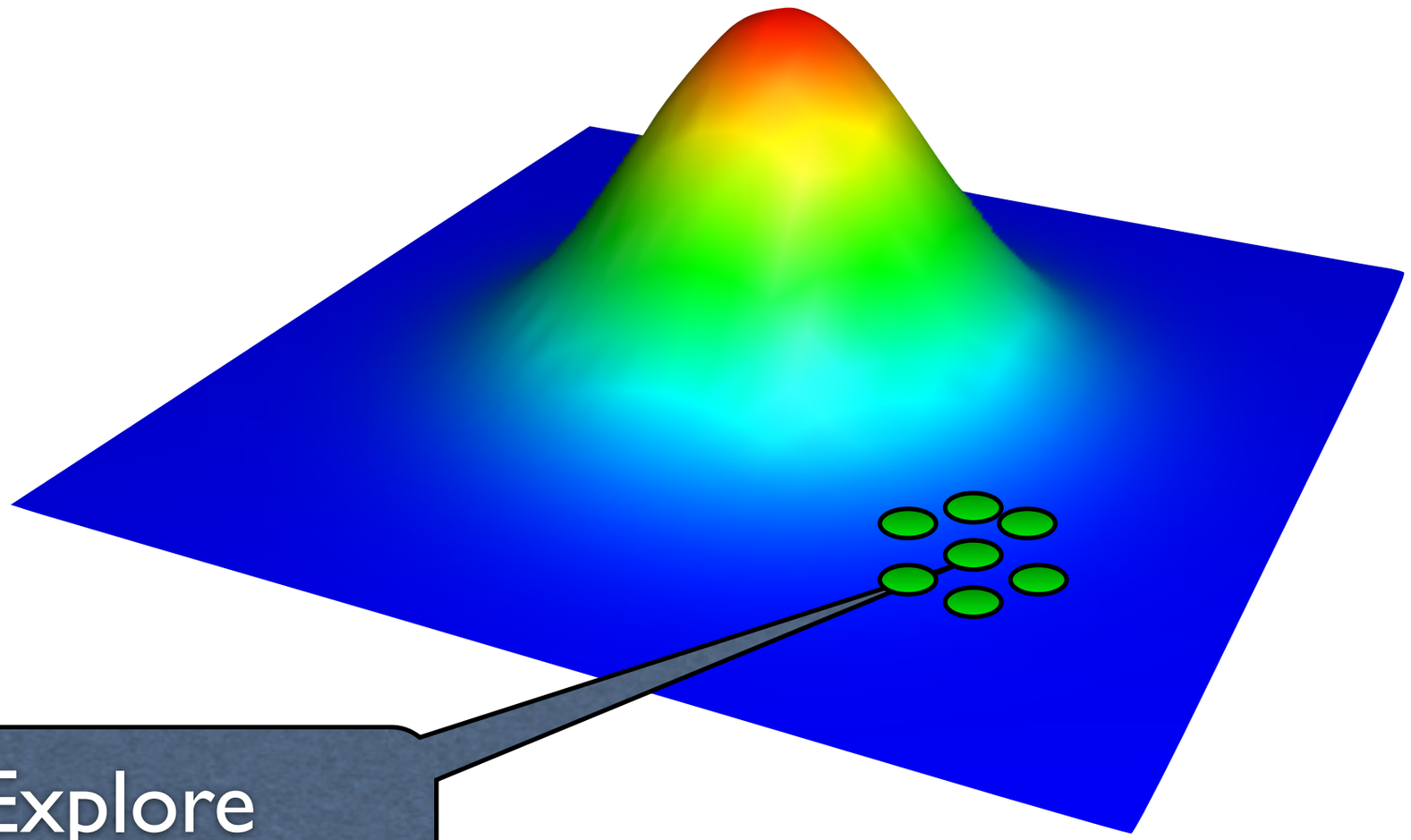
Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

Hill Climbing

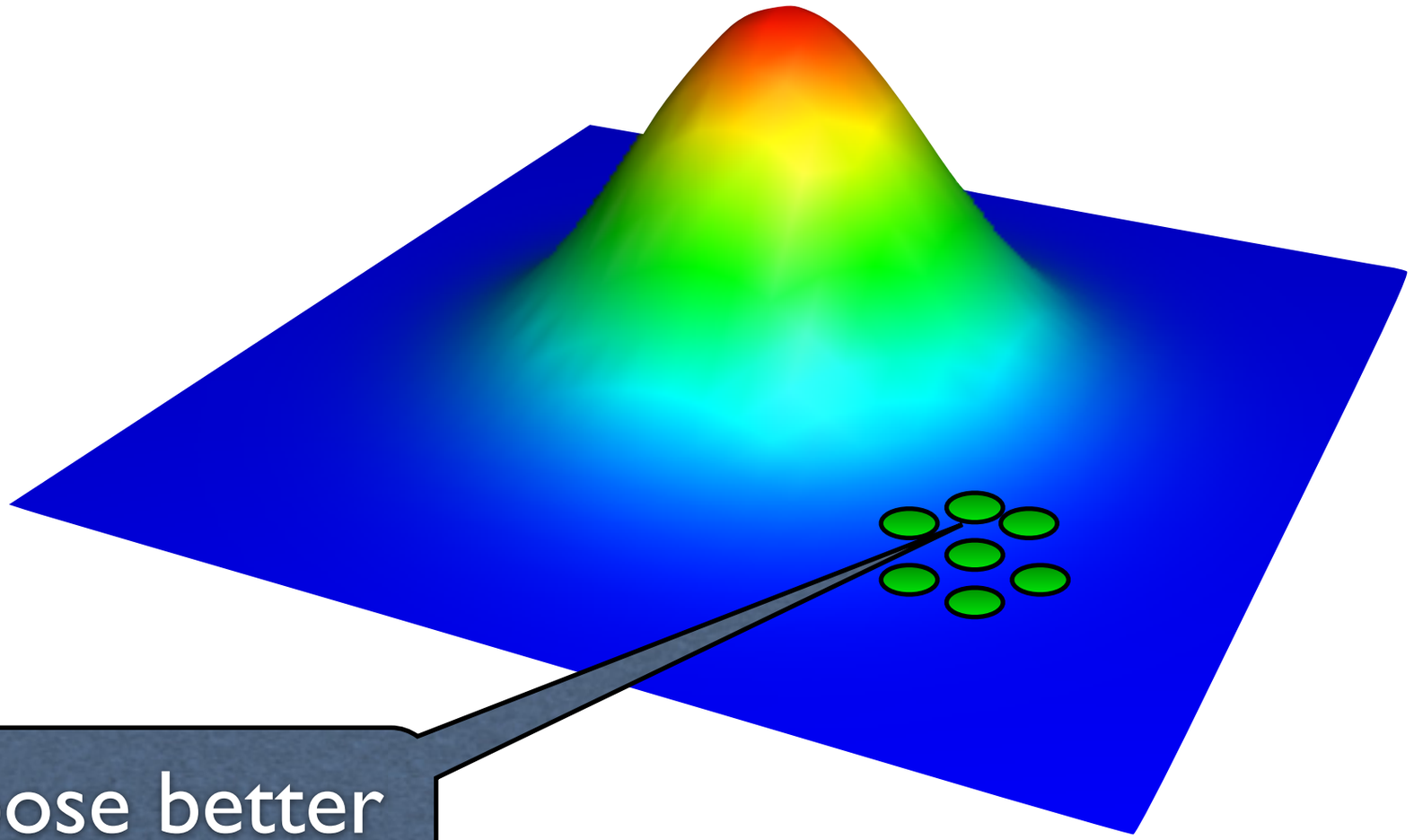


Hill Climbing



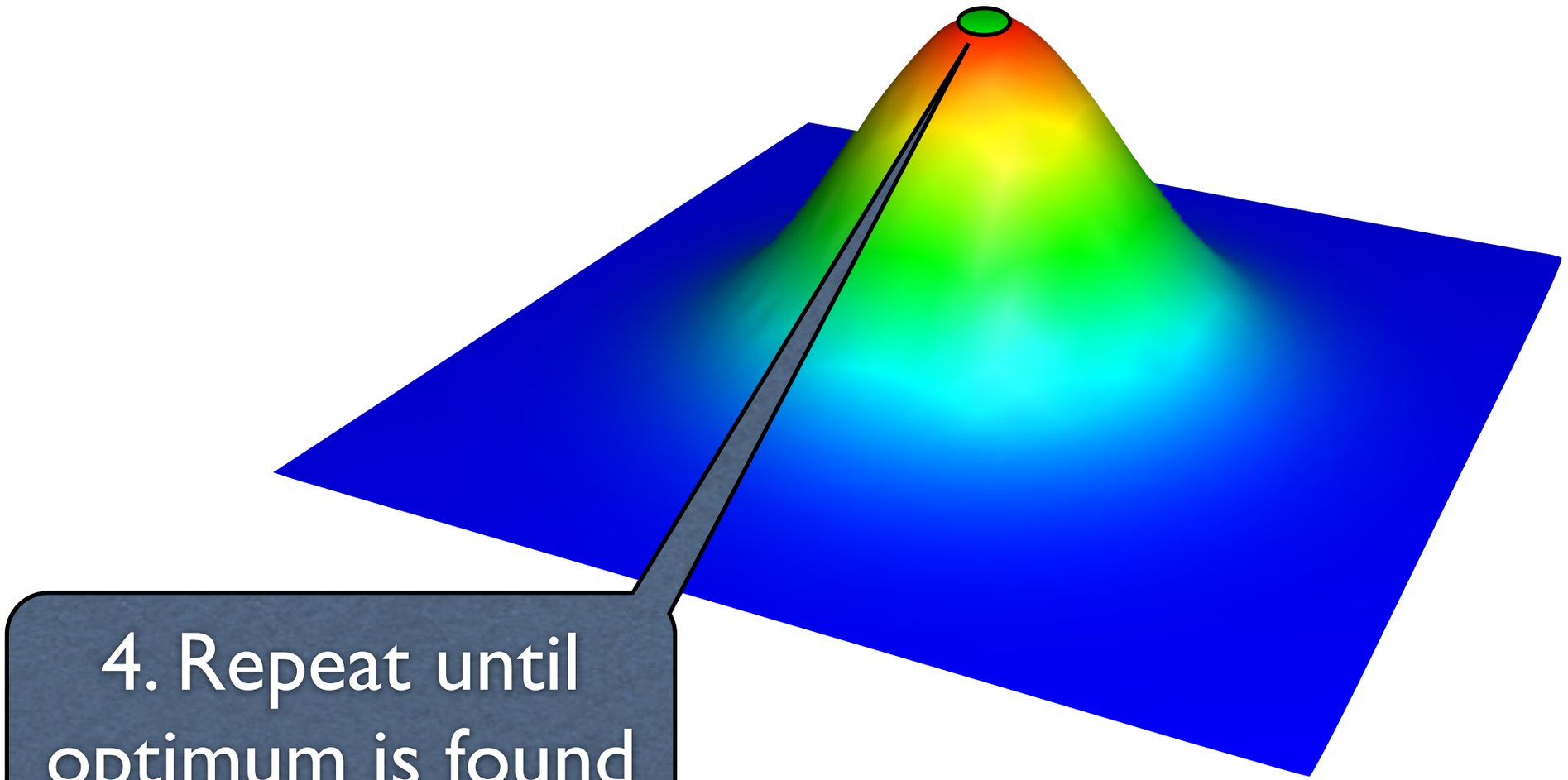
2. Explore Neighbourhood

Hill Climbing



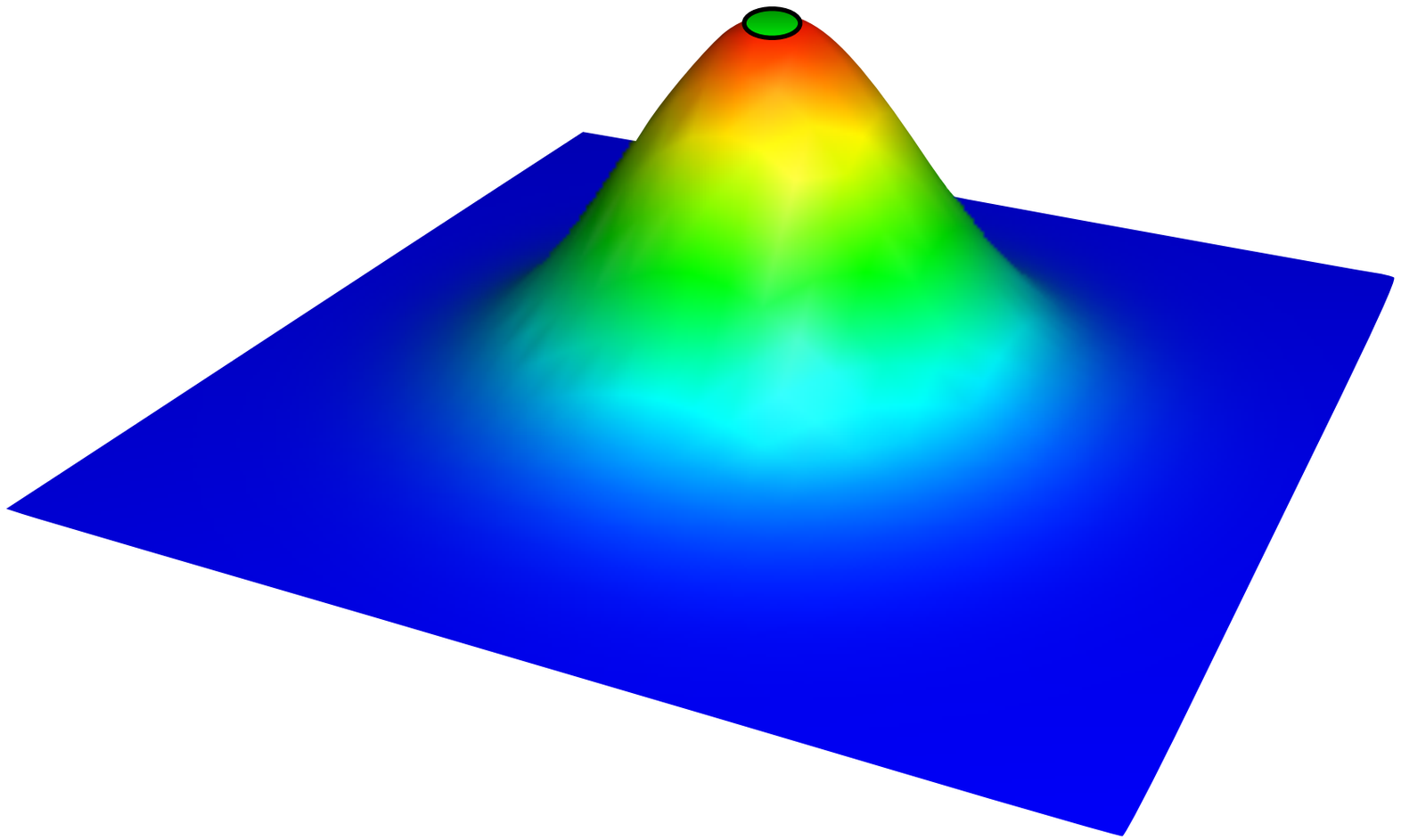
3. Choose better neighbour

Hill Climbing



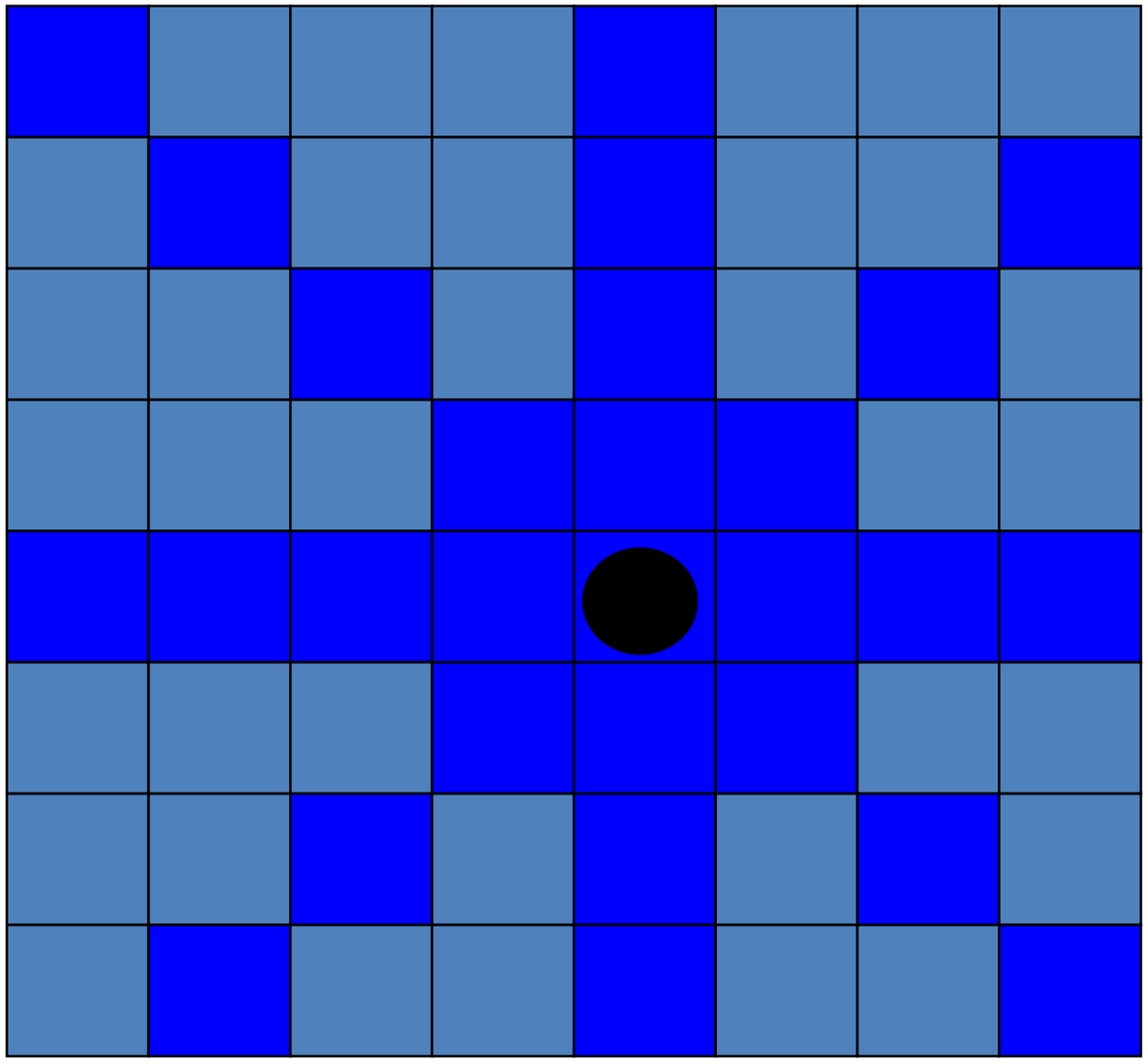
4. Repeat until optimum is found

Hill Climbing

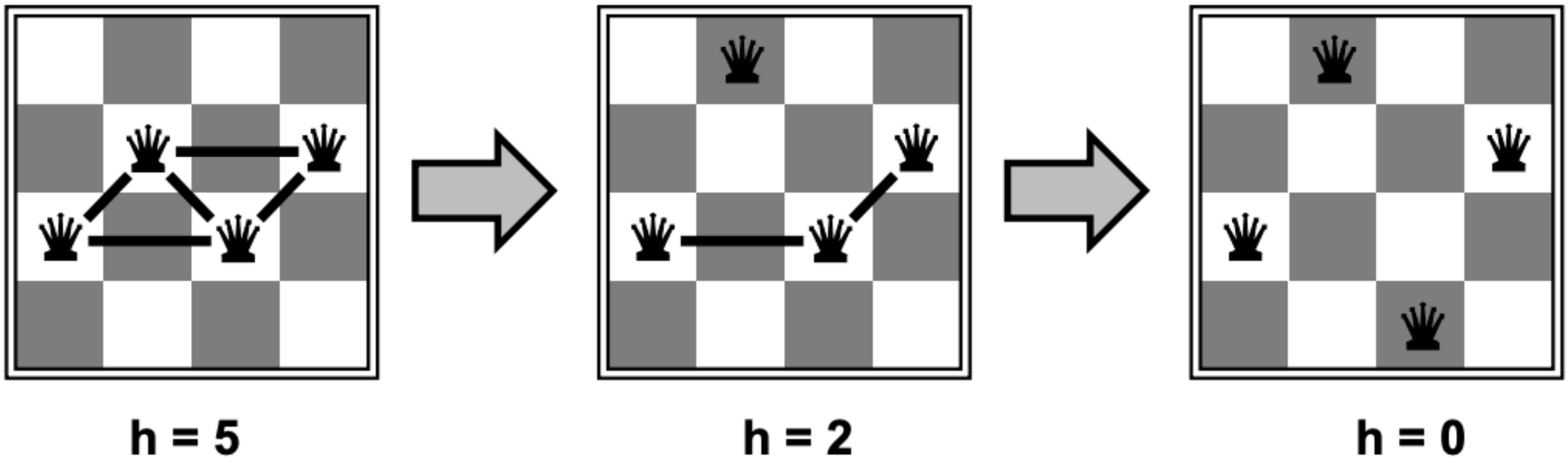


"Like climbing Everest in thick fog with amnesia"

Example: The 8-queens problem

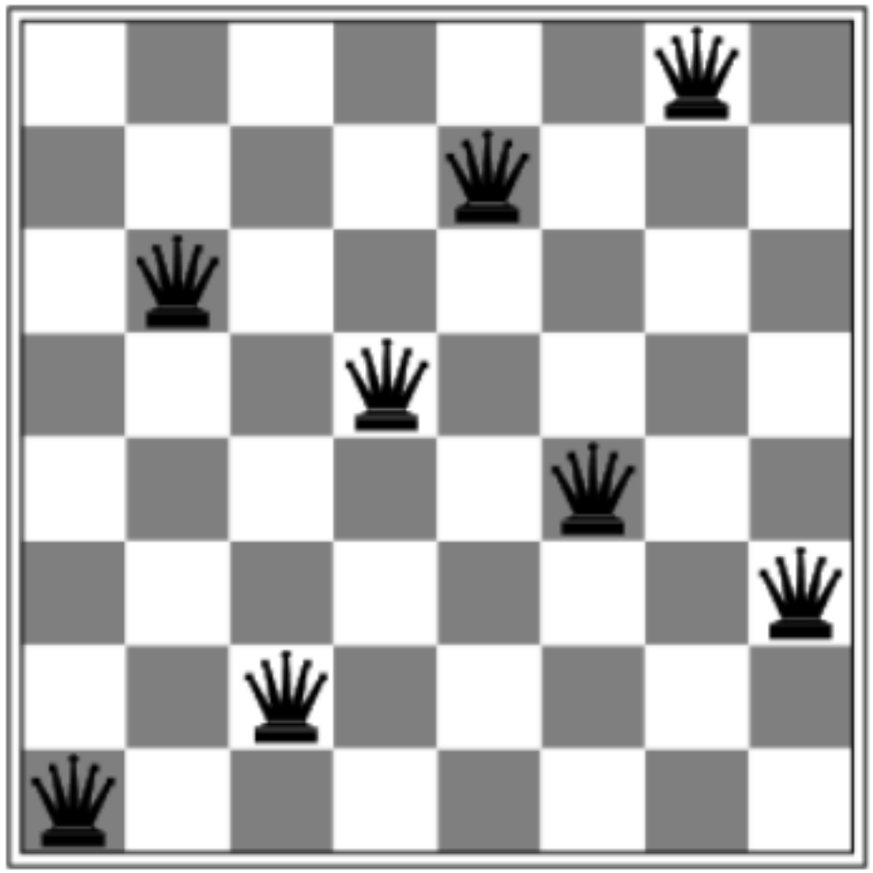


Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other



Heuristic h: number of ‘attacks’

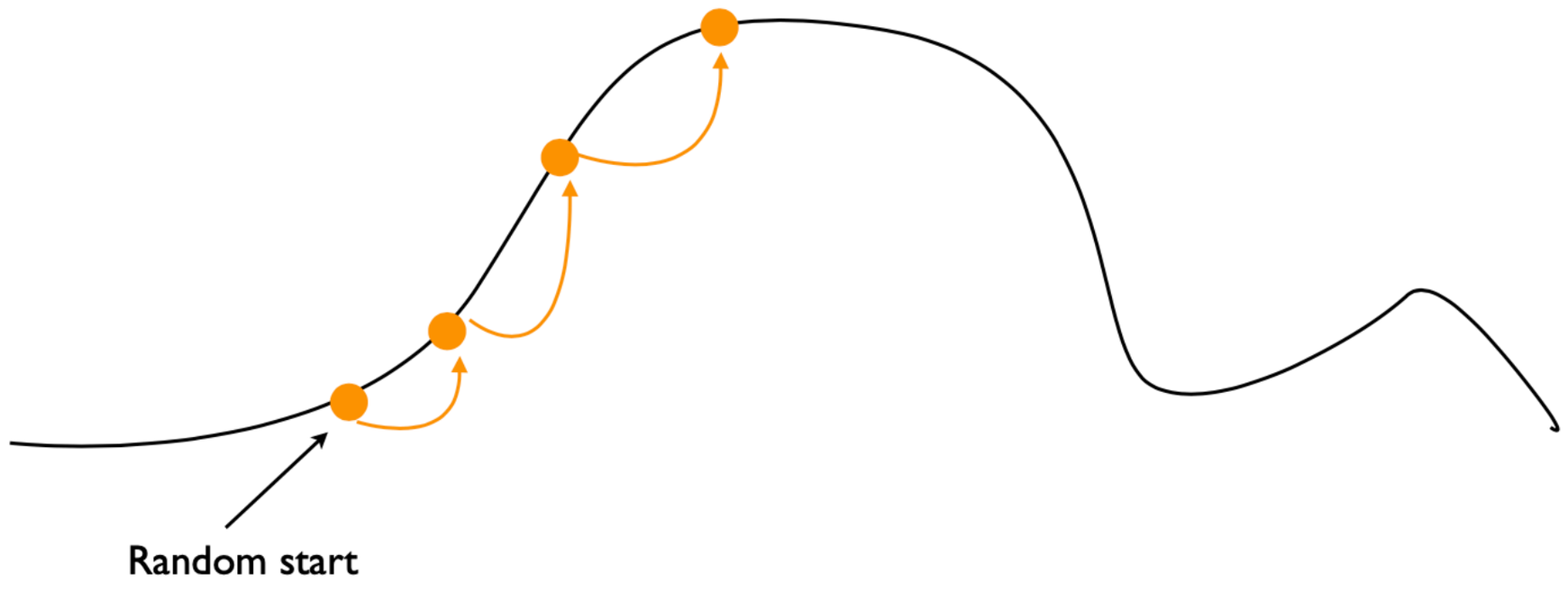
Local Optima



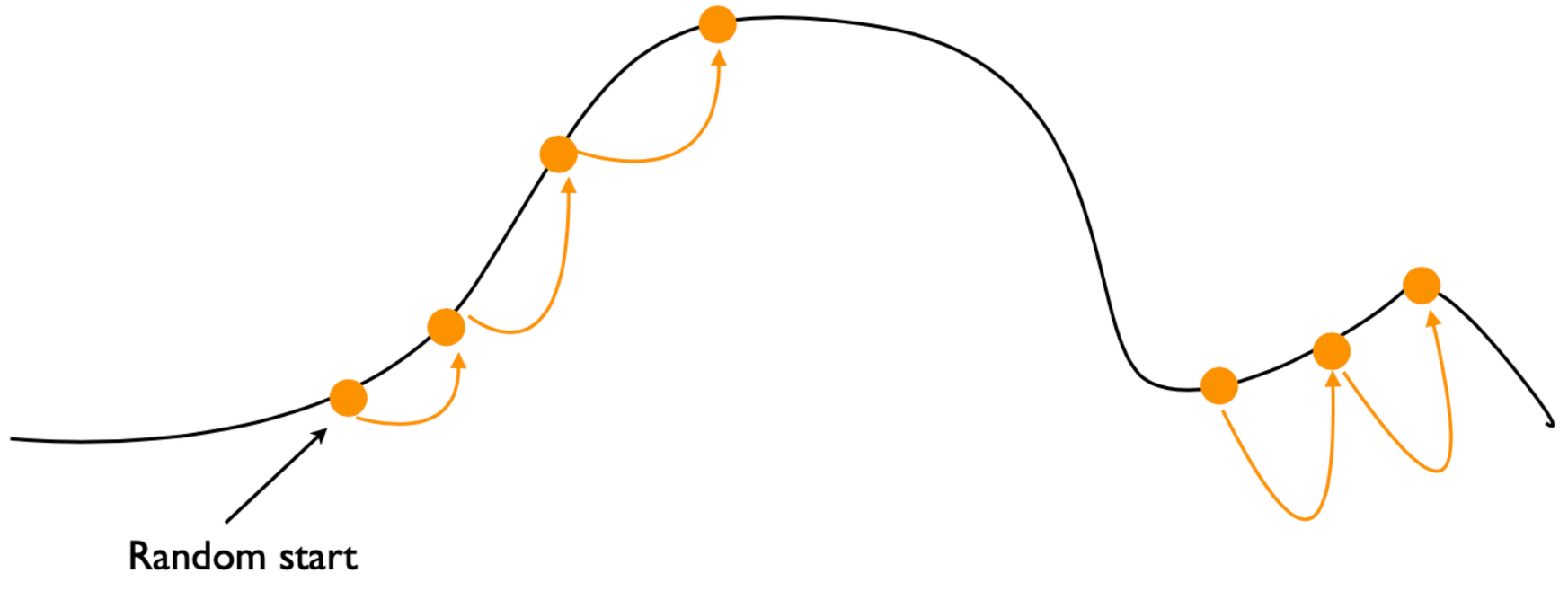
Fitness = 1

Check col. 4 and 7, white diagonal - and every change will create a worse state

Local Search



Restart



Hill Climbing

- Operator *U*: Choose best neighbour
- Gets stuck in local optima, unless fitness landscape is unimodal.
- Gets stuck on plateaux unless we also admit solutions with equal fitness.
- If search gets stuck: Random Restart
- Variations of *U*:
  - Choose first neighbour with better fitness
  - Randomised iterative improvements: Worsening moves accepted with certain probability
  - Probabilistic Hill Climber: Next search point *x'* in *V(x)* is selected with probability *p(x')* that is proportional to the fitness *f(x')*.
  - If *f(x)* is positive for all *x* in the search space then, for a maximisation problem, we can define *p(x)* as follows:
    $p(x') = f(x') / \Sigma_{(y \text{ in } V(x))} f(y)$

Ascent Strategy

- Not possible to know which one is better.
- If the current solution is near a local optimum, slowing down the ascent may (or may not) be better.
- Regardless of strategy, hill climbing monotonically climbs, until it reaches local/global optimum; it never goes down.
- Pros: cheap (fewer fitness evaluations compared to global search algorithms), easy to implement, repeated applications can give insights into the landscape, suitable for solutions that need to be built through small incremental changes
- Cons: more likely to get stuck in local optima, unable to escape local optima

Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

Tabu Search

- Proposed 1986 by F. Glover
- Starting from arbitrary solution
- Search operator selects x_{n+1} in $V(x_n)$ as the *non-tabu* x_{n+1} that locally optimises fitness
- Choice is independent of the fitness of the current solution $f(x_n)$.
  (If there is more than one candidate, one is chosen randomly)
- Tabu prevents re-exploration of already visited parts of search space
- Tabu attribute is not permanent (usually a certain number of iterations)
- Tabu implemented as a list to enumerate forbidden points and movements.
- Tabu list is continuously updated

Tabu Search

```
TabuSearch()
(1)     s ← s_0
(2)     s_best ← s
(3)     T ← [] // tabu list
(4)     while not stoppingCondition()
(5)         c_best ← null
(6)         foreach c ∈ GetNeighbours(s)
(7)             if (c ∉ T) ∧ (F(c) > F(c_best)) then c_best ← c
(8)         s ← c_best
(9)         if F(c_best) > F(s_best) then s_best ← c_best
(10)        append(T, c_best)
(11)        if |T| > maxTabuSize then removeAt(T, 0)
(12)    return sBest
```

Tabu Search

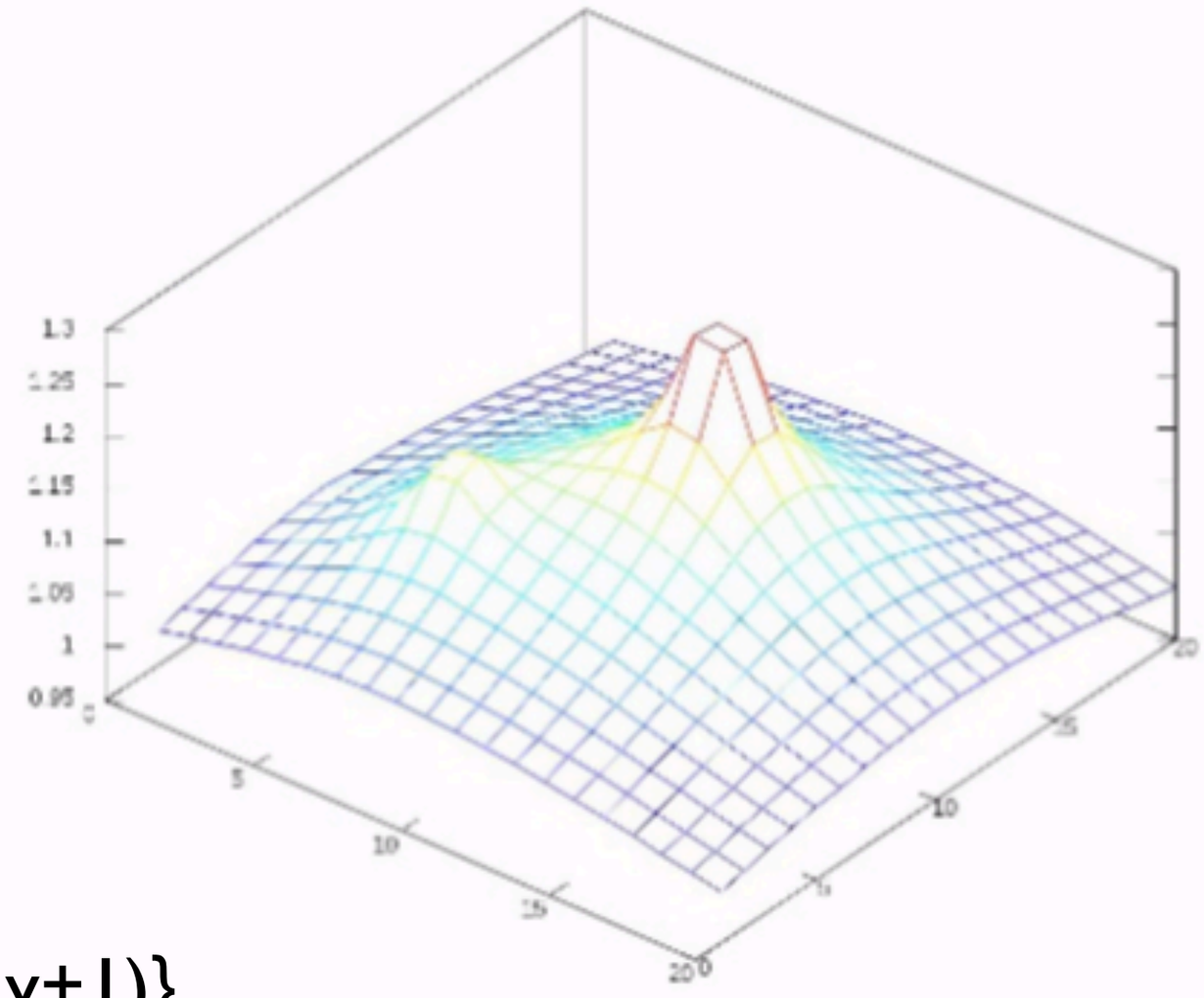
- Example objective function f(x, y)

- Search space a subspace of $\mathbb{Z}^2$:
  S = {0,1,…,19} x {0,1,…19}

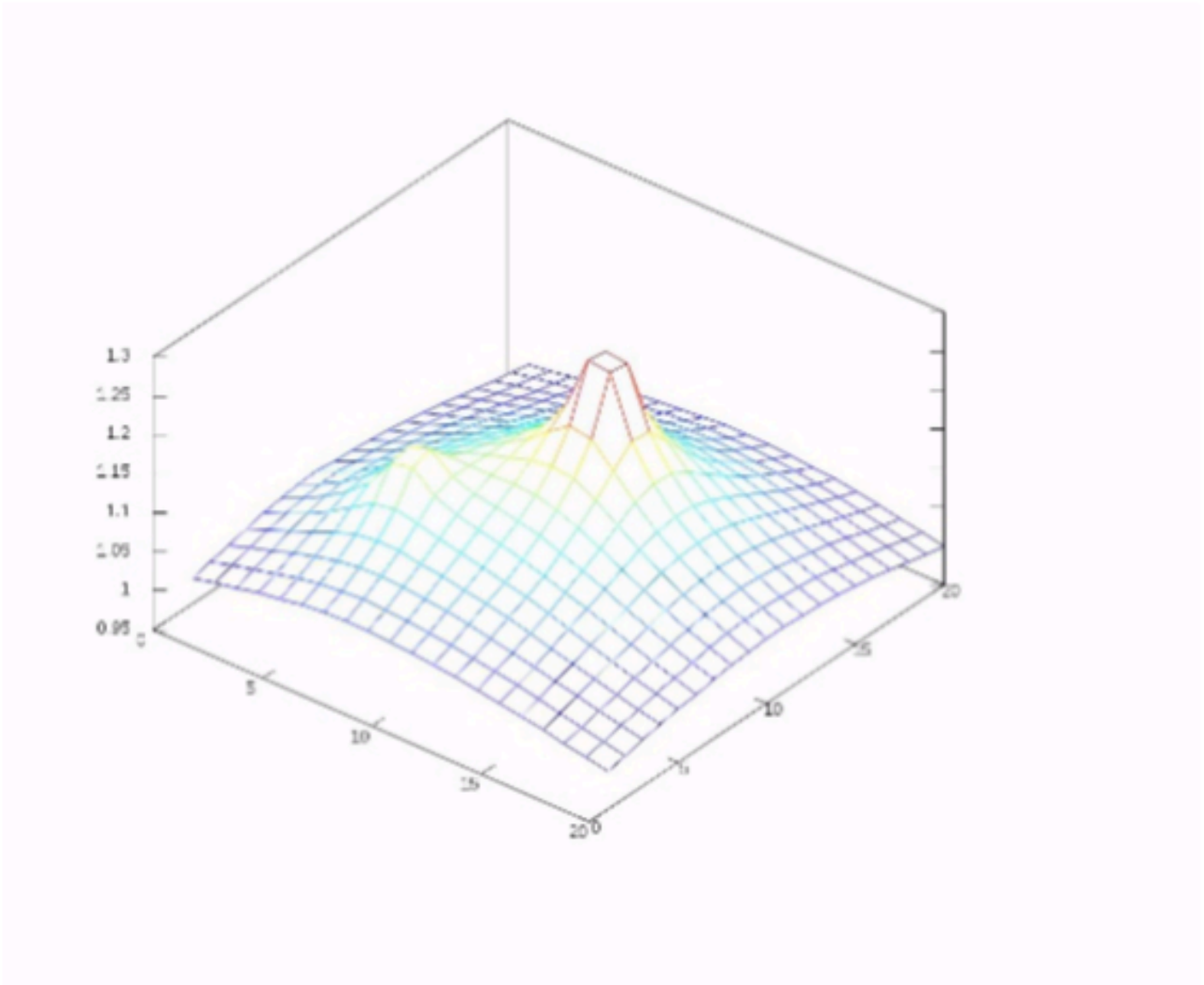
- Global optimum at P1=(10,10)

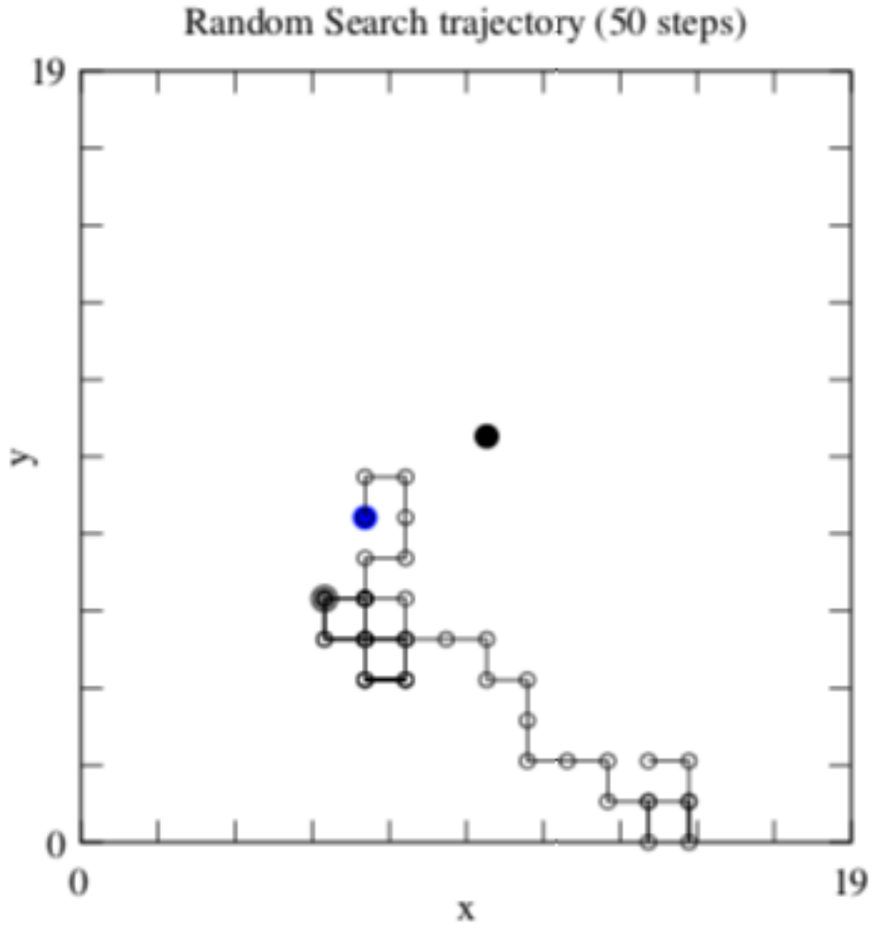
- Local maximum at P2=(6,6)

- Neighbourhood V(x,y) =
  V(x,y)={(x-1,y), (x+1,y), (x,y-1), (x,y+1)}



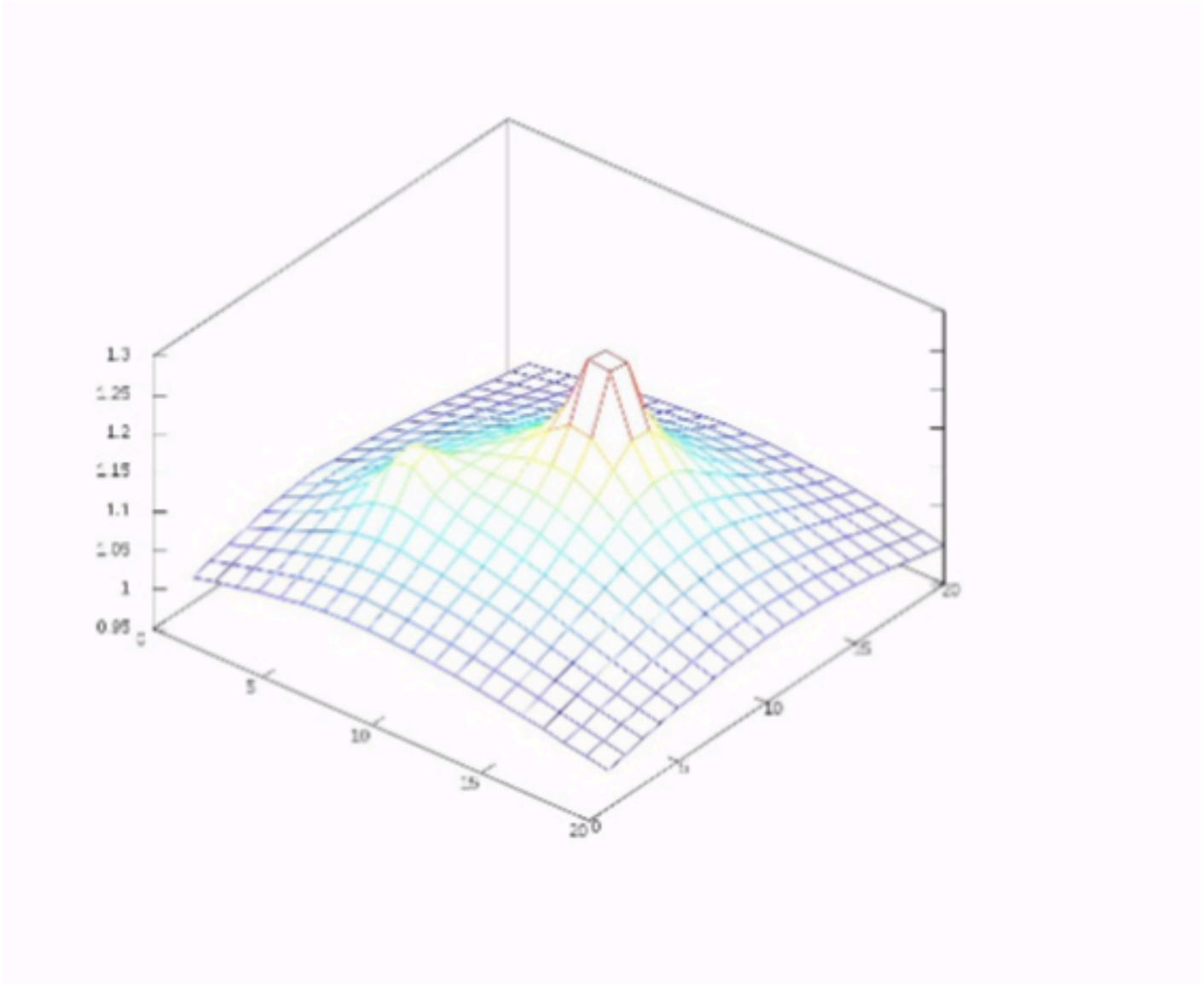
Recall: Random Walk

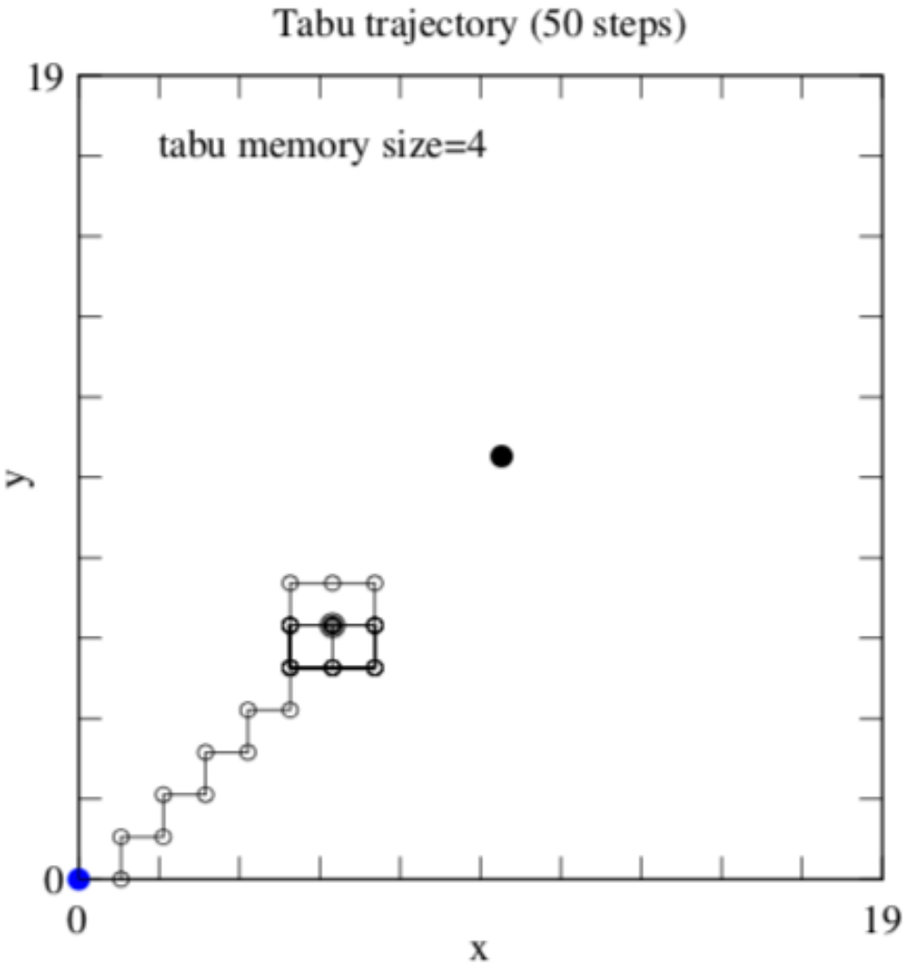




Random walk of 50 iterations, starting at (7,8)

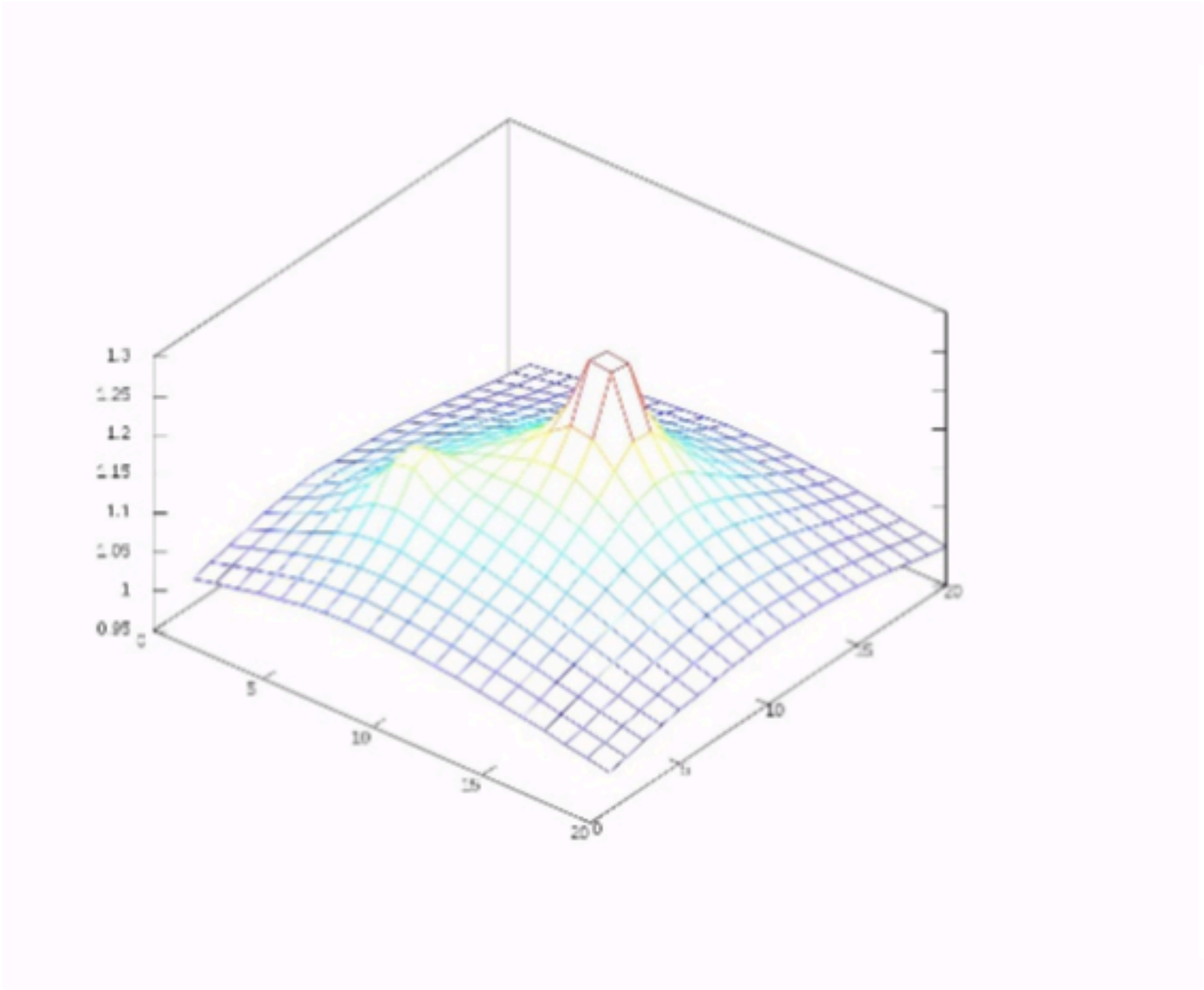
Tabu Search

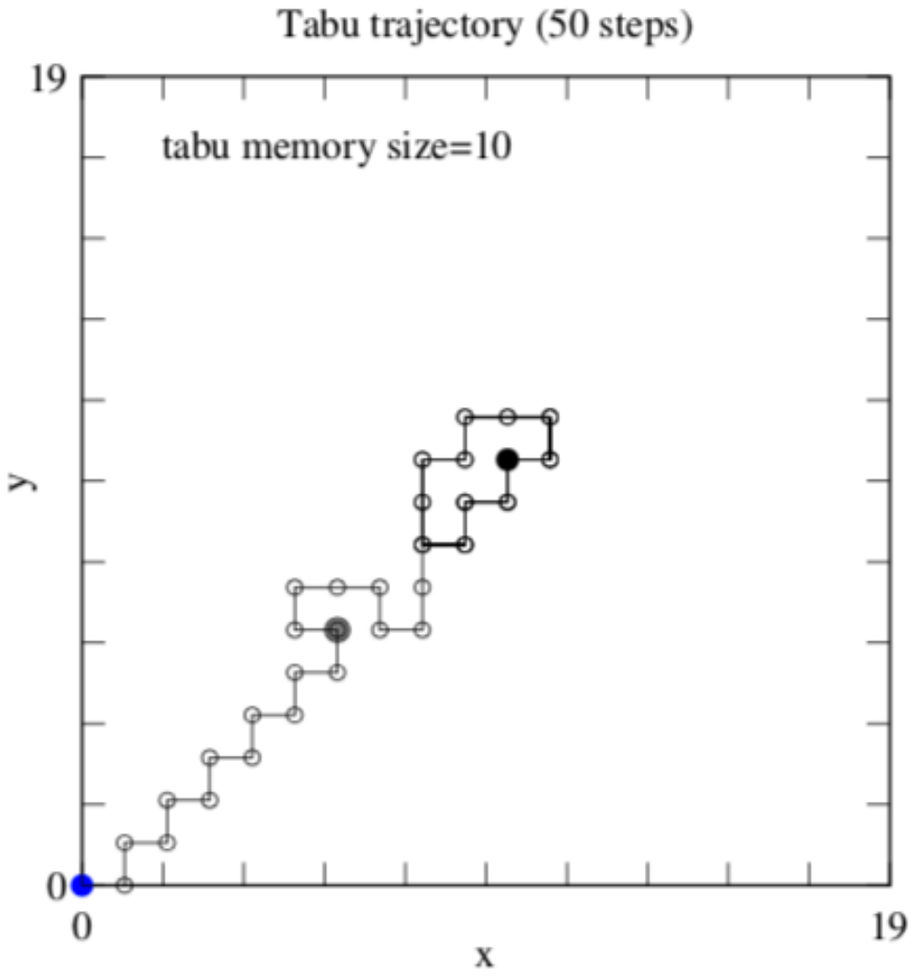




Tabu Search, 50 steps, tabu size 4

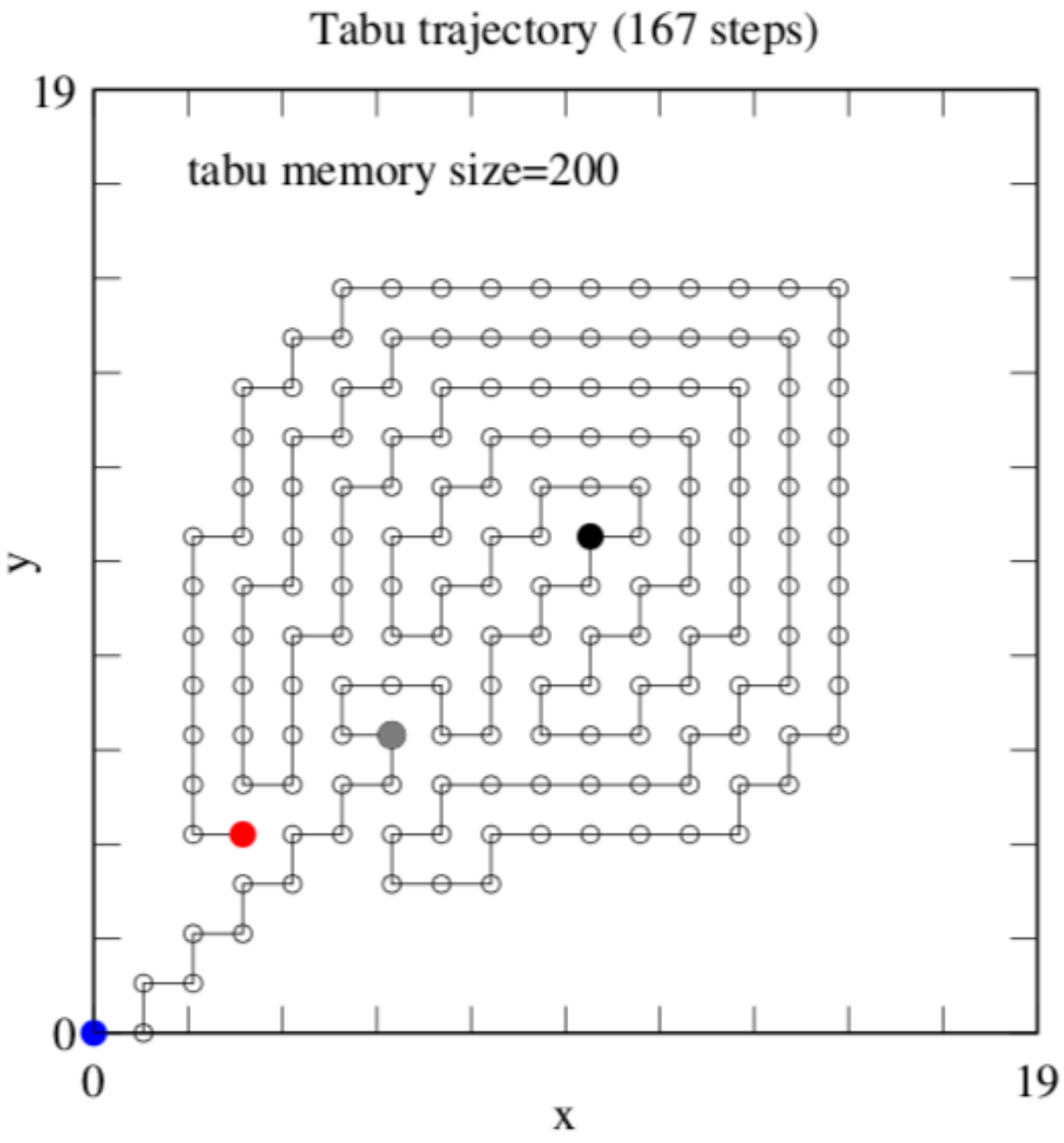
Tabu Search



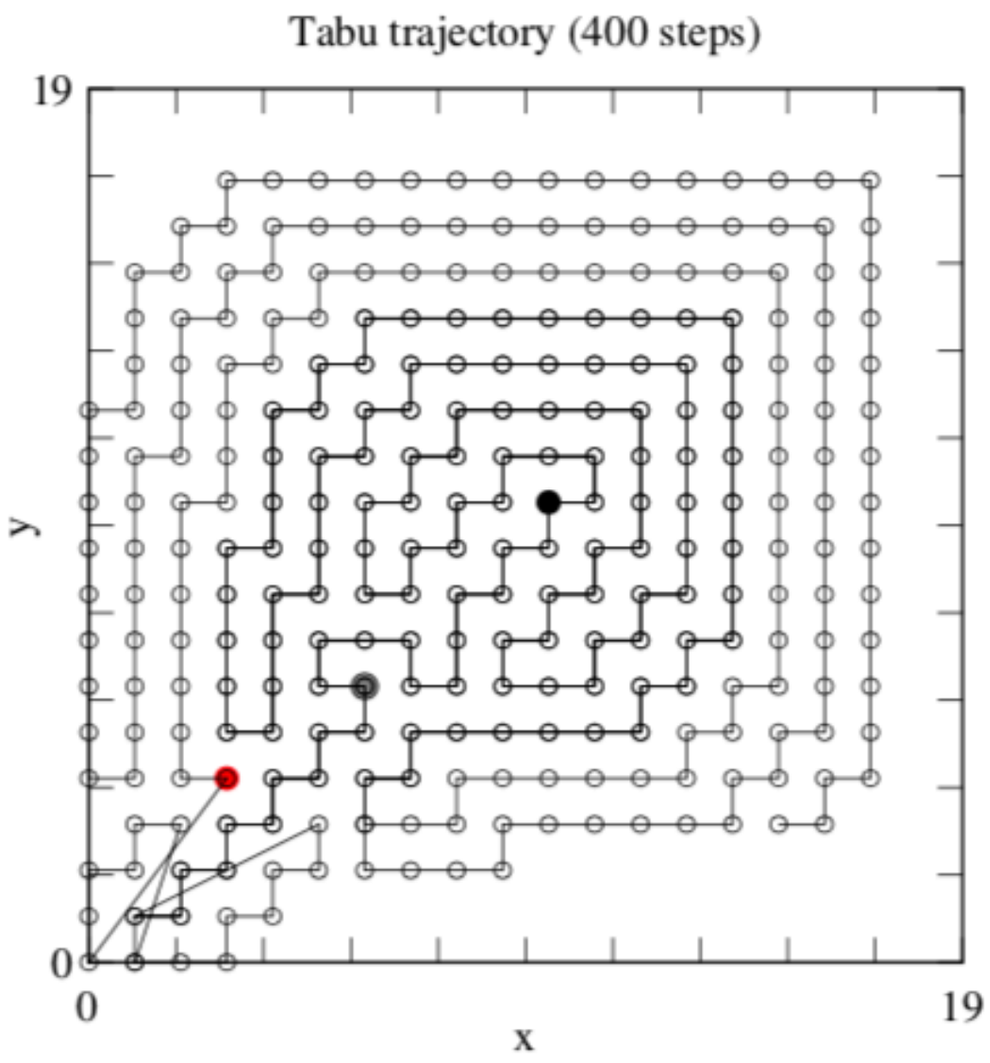


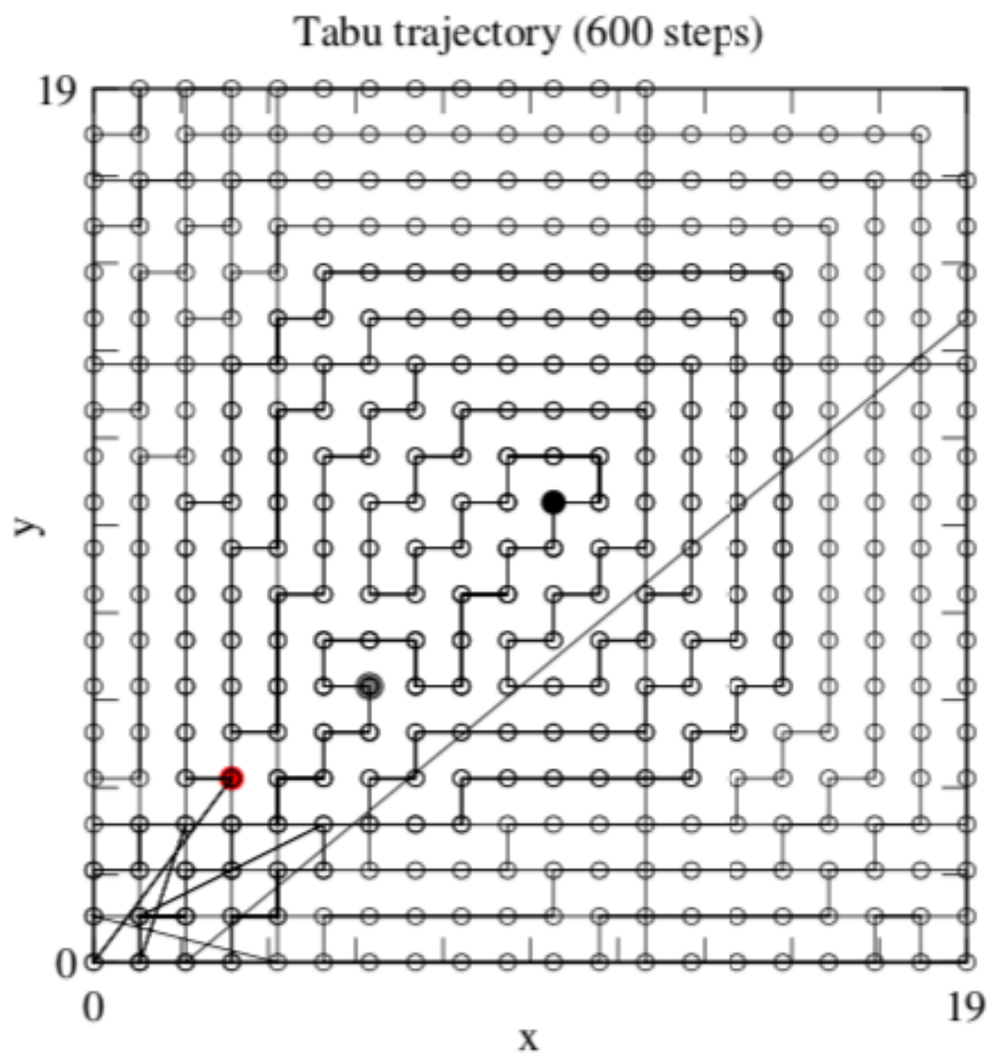
Tabu Search, 50 steps, tabu size 10

Tabu Search

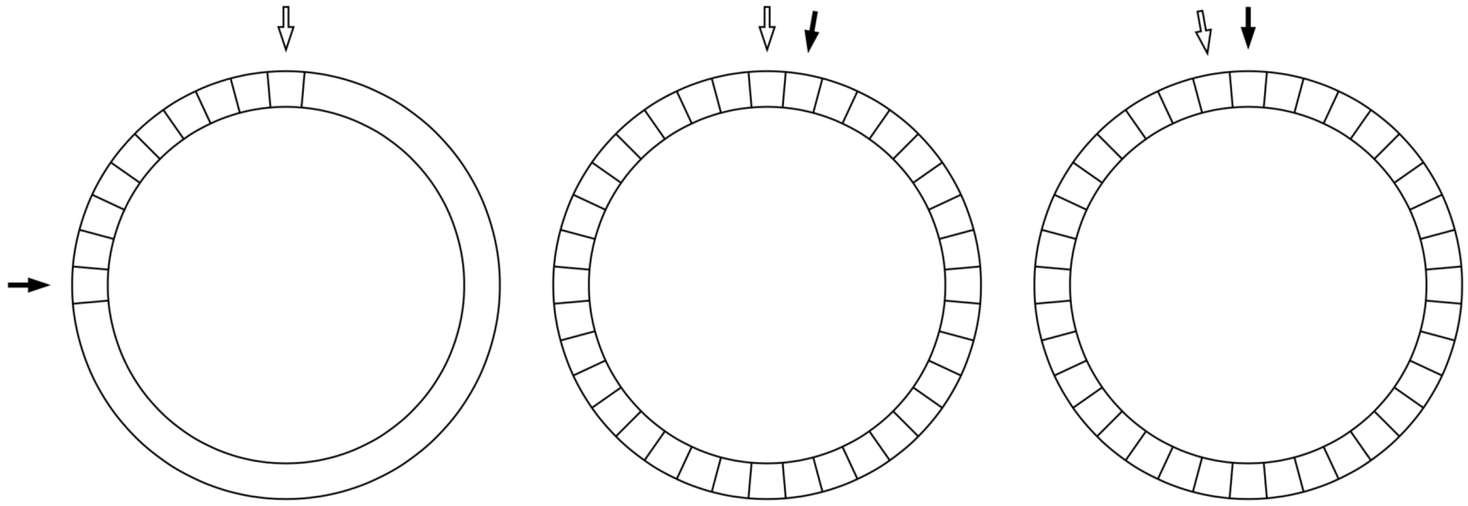


Tabu Search


Tabu trajectory (400 steps)
19
0
0
19
x
y


Tabu trajectory (600 steps)
19
0
0
19
x
y

Tabu Search



- Obvious implementation of tabu list: Circular buffer
- Often it is more convenient to store items that are related to the visited solutions, rather than the solutions themselves, e.g.
  - Attributes belonging to the visited solutions such as fitness values
  - Moves that have been used to explore the search space.
- Alternative: Banning time / tabu tenure:
  - If a move m has been used in iteration t, or the attribute h characterises the current solution at time step t, then the inverse move of m, or the attribute h, will be forbidden until iteration t+k, where k is the banning time
  - k is often chosen randomly in each iteration according to a specific probability distribution

Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

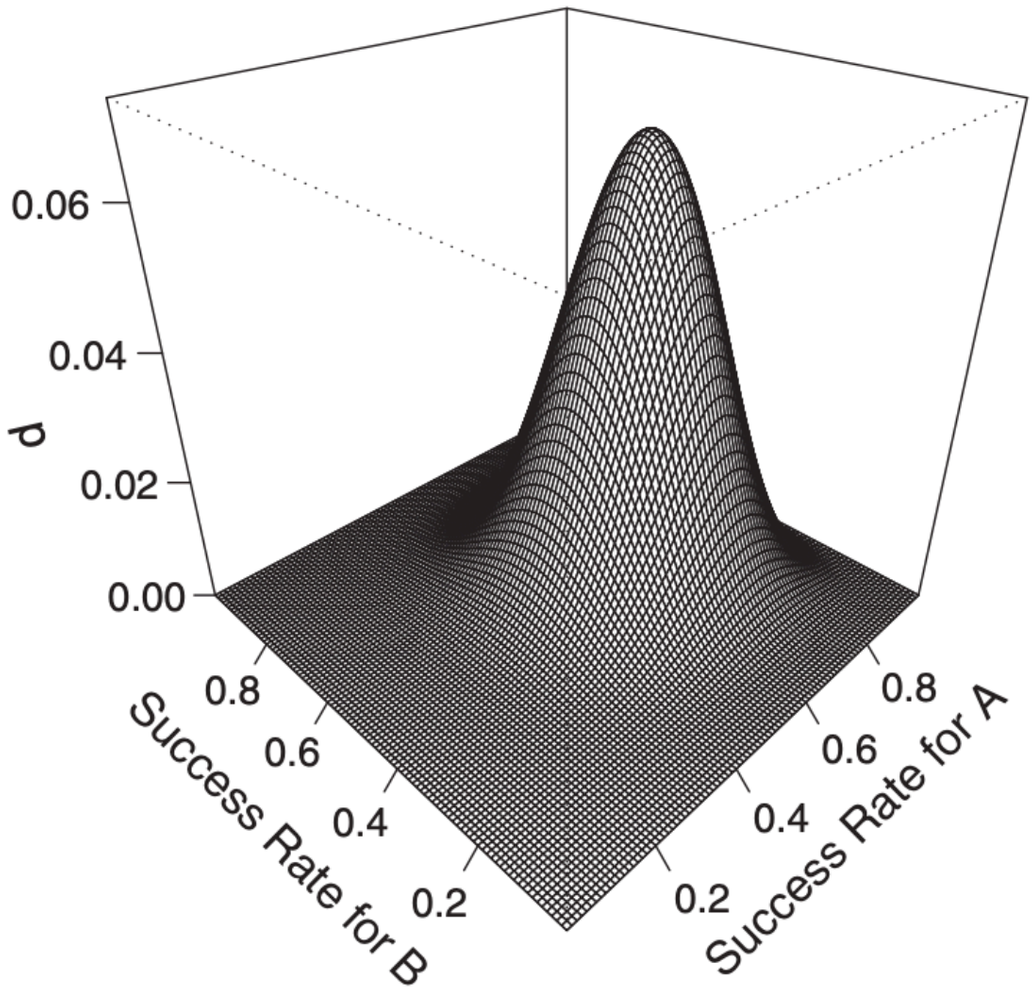
Motivation

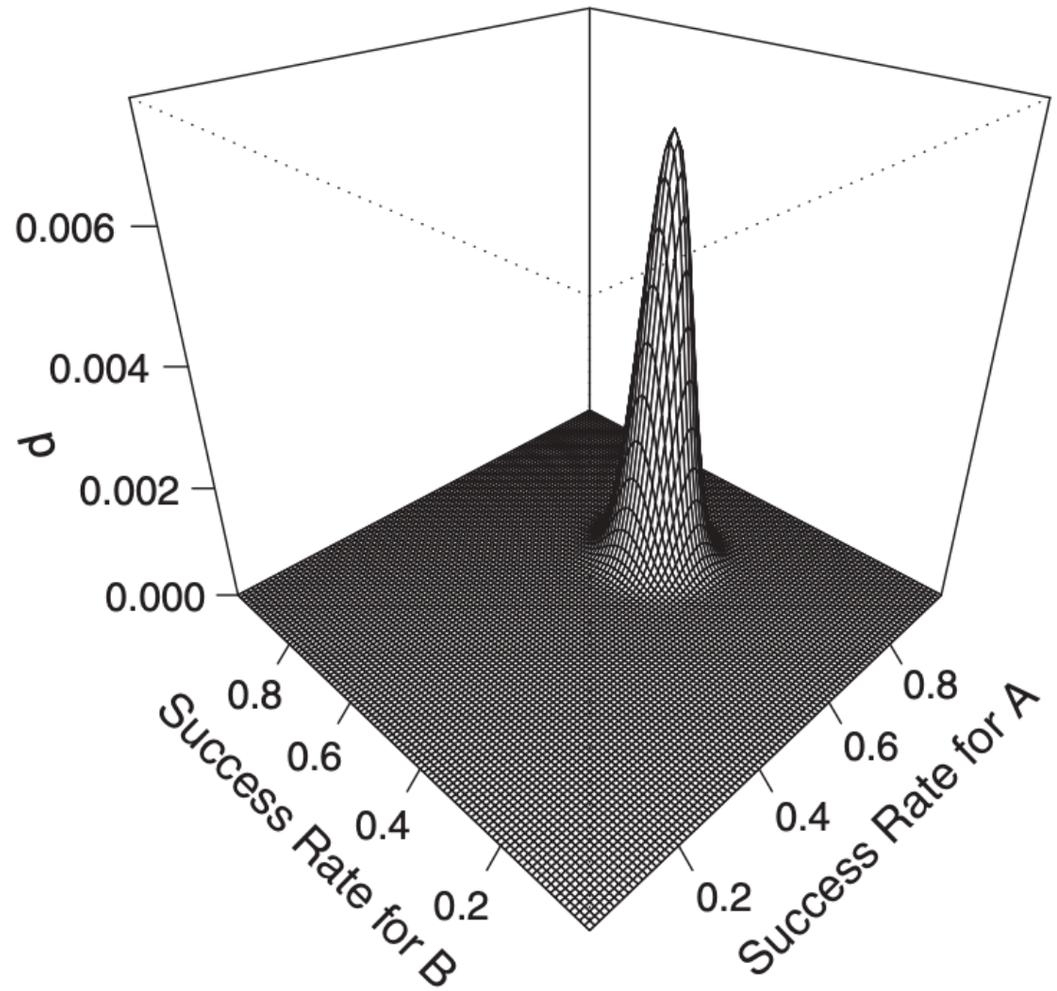
- A randomised algorithm may be strongly affected by chance:
  - It may find an optimal solution quickly or may never converge towards an acceptable solution.
  - Running a randomised algorithm twice on the same instance of a software engineering problem usually produces different results.
- How do we evaluate the effectiveness of search-based software engineering techniques?
  - Measurements: Effectiveness, Performance, Time to convergence, Robustness, Diversity, etc
- Important: We need to study the probability distribution of these metrics
  - Run the algorithm several times in an independent way, and then collect data about its results and performance.
  - Looking at averages can be misleading, as randomised algorithms can yield very complex and high variance probability distributions.

Example

- Experiment:
  - Fault Finding, Crashing Android Apps
  - n = 10 runs
- Technique A: Random Testing
  - Success rate 70% (7 out of 10 runs found a crash)
- Technique B: Tabu Search
  - Success rate: 50% (5 out of 10 runs found a crash)
- Is A better than B?

Example





- Probabilities to obtain a = 0.7n and b = 0.5n when n = 10 (left) and n = 100 (right) for different success rates of the algorithms A and B

Statistical Difference

- When comparing algorithms A and B we need to decide on a measure M, e.g. code coverage or execution time.
- We run both A and B a large enough number (n) of times, in an independent way, to collect information on the probability distribution of M for each algorithm.
- Statistical test to assess whether there is enough empirical evidence to claim that there is a difference between the two algorithms (e.g. A is better than B).
- Null Hypothesis H0: There is no difference between A and B.
- The statistical test aims to verify whether one should reject the null hypothesis H0.
- What aspect of the probability distribution is being compared depends on the used statistical test.
    - For example, a t-test compares the mean values of two distributions, whereas other tests focus on the median or proportions

Type I and II Errors

- There are two possible types of error when performing statistical testing:
  - (I) H0 is rejected although it is true (i.e. claiming that there is a difference between two algorithms when actually there is none).
  - (II) H0 is accepted when it is false (i.e. claiming the two algorithms are equivalent when there is a difference).
- The p-value of a statistical test denotes the probability of a Type I error.
- The significance level **α** of a test is the highest p-value one accepts for rejecting H0.
  - A typical value, inherited from widespread practice in natural and social sciences, is **α** = 0.05.
  - The use of **α** = 0.05 is much debated

Type I and II Errors

One- vs Two-Tailed Tests

- In a one-tailed test, we make assumptions about the relative performance of the algorithms.
  - E.g. a new sophisticated algorithm A is better than a naive algorithm B used in the literature.
  - One would detect a statistically significant difference when A is indeed better than B, but ignore the 'unlikely' case of B being better than A
- In a two-tailed test, we reject H0 if the performance of A and B are different regardless of which one is the best.
- Historical example:
  - Checking whether there is the right percent of gold (carats) in coins.
  - A dishonest coiner might produce coins with lower percent of gold than declared, and so a one-tailed test would be used rather than a two-tailed.
  - It is unlikely that too much gold is used (and it wouldn't be a problem).
- Using a one-tailed test has the advantage, compared with using a two-tailed test, that the resulting p-value is lower (so it is easier to detect statistically significant differences)
- In software engineering there are hardly any cases where a one-tailed test would be used; two-tailed tests should be used.

Parametric vs. Non-Parametric

- A parametric test makes assumptions on the underlying distribution of the data.
  - E.g., the t-test assumes normality and equal variance of the two data samples.
- A nonparametric test makes no assumptions about the distribution of the data.
- Nonparametric tests are usually less powerful than parametric ones when the latter's assumptions are fulfilled.
  - Mann-Whitney U Test / Wilcoxon Rank-Sum Test
- The assumptions of the t-test are, in general, not met.
- In general, one cannot know how many data points (n) one needs to reach reliable results.
  - A rule of thumb is to have at least n = 30 for each data sample
  - This rule of thumb originates from behavioural science, we don't really know if it is effective for randomised algorithms in software engineering

Tests vs. Deterministic Algorithms

- So far we assumed that both algorithms A and B are randomised.
- If one of them is deterministic, it is still important to use statistical testing.
  - Use the nonparametric one-sample Wilcoxon test
  - Given m_B (the performance measure of the deterministic algorithm B), a one-sample Wilcoxon test would verify whether the performance of A is symmetric around m_B
    - That is, whether by using A, one is as likely to obtain a value lower than m_B as otherwise.

Effect Size

- Given a large enough number of runs n, it is most of the time possible to obtain statistically significant results with a t-test or U-test.
    - Two different algorithms are extremely unlikely to have exactly the same probability distribution.
    - With a large enough n, one can obtain statistically significant differences even if they are so small as to be of no practical value.
- Therefore, in addition to statistical difference, it is crucial to assess the magnitude of the improvement, i.e. the *effect size*.
- Effect sizes can be divided in two groups:
    - Unstandardised effect sizes are dependent on the unit of measurement.
    - Standardised are independent of the unit of measurement.

Effect Size

- Consider the difference in means between two algorithms $\Delta = \mu^A - \mu^B$
  - This value Δ has a measurement unit, that of A and B.
  - E.g. expected number of test executions to find the first failure.
- Testing artefact $\Delta_1 = \mu^A - \mu^B = 100 - 1 = 99$
- Testing artefact $\Delta_2 = \mu^A - \mu^B = 100\,000 - 200\,000 = -100\,000$
- Deciding on the basis of Δ_1 and Δ_2 which algorithm is better is difficult because the two scales of measurement are different.
- Empirical analyses of randomised algorithms, if they are to be reliable and generalisable, require the use of large numbers of artefacts (e.g. programs).
  - The complexity of these artefacts is likely to widely vary, such as the number of test cases required to fulfil a coverage criterion on various programs.
- The use of standardised effect sizes is therefore necessary to be able to compare results across artefacts and experiments.

Effect Size

- The most known effect size is the so-called d family:
  - $d = (\mu^A - \mu^B) / \sigma$
  - I.e., the difference in means is scaled over the standard deviation.
- Problem: Assumes normality of the data, and strong departures can make it meaningless.
- In this case, a nonparametric effect size should be preferred
- Vargha and Delaney's $\hat{A}_{12}$ statistic:
  - Given a performance measure M, $\hat{A}_{12}$ measures the probability that running algorithm A yields higher M values than running another algorithm B.
  - If the two algorithms are equivalent, then $\hat{A}_{12} = 0.5$.
  - This effect size is easier to interpret compared with the d family.
  - $\hat{A}_{12} = 0.7$ entails one would obtain better results 70% of the time with A.

Vargha Delaney $\hat{A}_{12}$

- Although this type of nonparametric effect size is not common in statistical tools, it can be very easily computed:
  - $\hat{A}_{12} = (R_1/m - (m + 1) / 2) / n$
- m is the number of observations in the first data sample, whereas n is the number of observations in the second data sample. (Usually m = n)
- R_1 is the rank sum of the first data group under comparison
  - For example, assume the data X = { 42, 11, 7 } and Y = { 1, 20, 5 }.
  - The data set X would have ranks { 6, 4, 3 }, whose sum is 13, whereas Y would have ranks { 1, 5, 2 }.
- The rank sum is a basic component in the Mann–Whitney U-test, and most statistical tools provide it.

Odds Ratio

- For dichotomous results, the odds ratio is the most used effect size measure
- It is a measure of how many times greater the odds are that a member of a certain population will fall into a certain category than the odds are that a member of another population will fall into that category
- Given a the number of times algorithm A finds an optimal solution, and b for algorithm B, the odds ratio is calculated as

$$\psi = \frac{a+\rho}{n+\rho-a} / \frac{b+\rho}{n+\rho-b},$$

- where ρ is any arbitrary positive constant (e.g. ρ = 0.5) used to avoid problems with zero occurrences.
- There is no difference between the two algorithms when ψ = 1.
- The cases in which ψ >1 imply that algorithm A has higher chances of success.

Confidence Intervals

- Both $\hat{A}_{12}$ and Ψ are standardised effect size measures.
- As their calculation is based on a finite number of observations (n), they are only estimates of the real $\hat{A}_{12}$ and Ψ.
    - If n is low, these estimations might be very inaccurate.
- One way to deal with this problem is to calculate CIs for them.
- A (1 - α) CI is a set of values for which there is (1 - α) probability that the value of the effect size lies in that range.
- For example, if one has $\hat{A}_{12} = 0.54$ and a (1 - α) CI with range [0.49, 59], then with probability (1 - α) the real value $\hat{A}_{12}$ lies in [0.49, 59] (where $\hat{A}_{12} = 0.54$ is its most likely estimation).
- Such effect size CIs can facilitate decision making as they enable the comparison of the costs of alternative algorithms while accounting for uncertainty in their estimates.

Recommended Reading

Andrea Arcuri and Lionel Briand. "A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering." Software Testing, Verification and Reliability 24.3 (2014): 219-250.

Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing

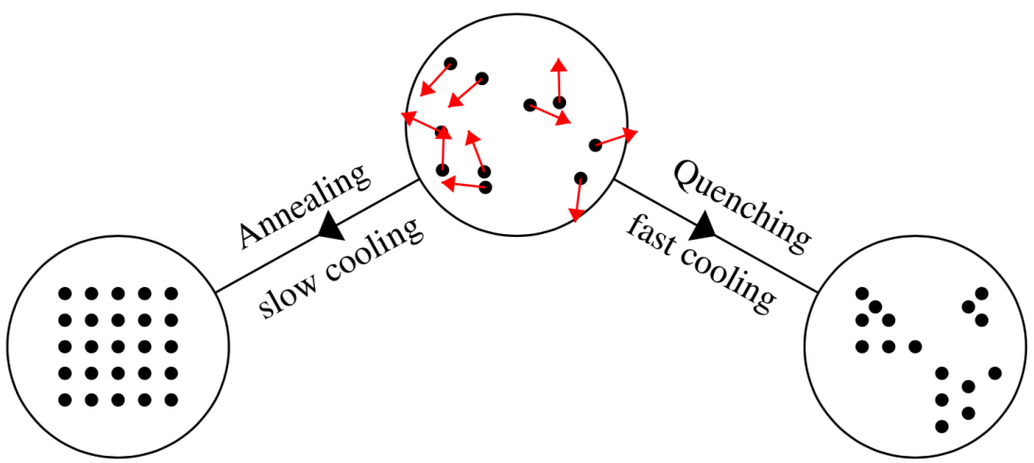
Simulated Annealing

- Metallurgy: In heated metal atoms undergo disordered movements of large amplitude.

- Cooling the metal down progressively reduces movement and stabilises atoms around fixed positions in a regular crystal structure with minimal energy.

- In this state ductility is improved and the metal becomes easier to work.

- Alternative: Quenching, rapidly cooling down metal. Quenching causes metal to be more fragile, but also harder and more resistant to wear and vibration

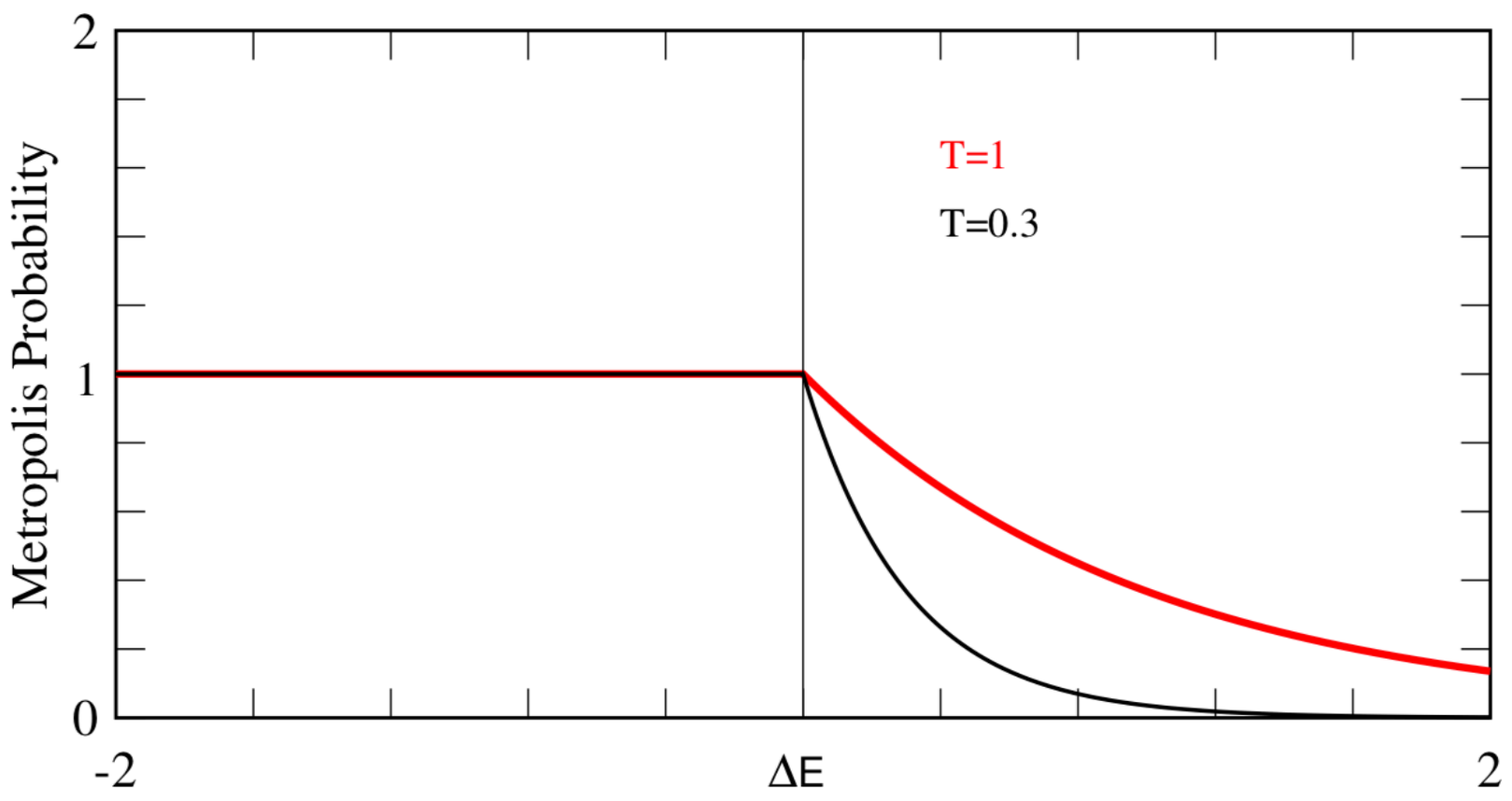


Simulated Annealing

- Algorithm searches for the minimum of a given objective function, called energy E.
- Initialised with arbitrary admissible solution.
- Initial temperature
- Elementary transformations: Algorithm does not test all neighbours, but selects a random move among the allowed ones.
- If the move leads to a lower energy value, then the new configuration is accepted and becomes the current solution.
- Even moves that lead to an increase of the energy can be accepted with positive probability.
- The probability of accepting solutions that worsen the fitness are computed from the energy variation ΔE before and after the given move: $\Delta E = E_{new} - E_{current}$

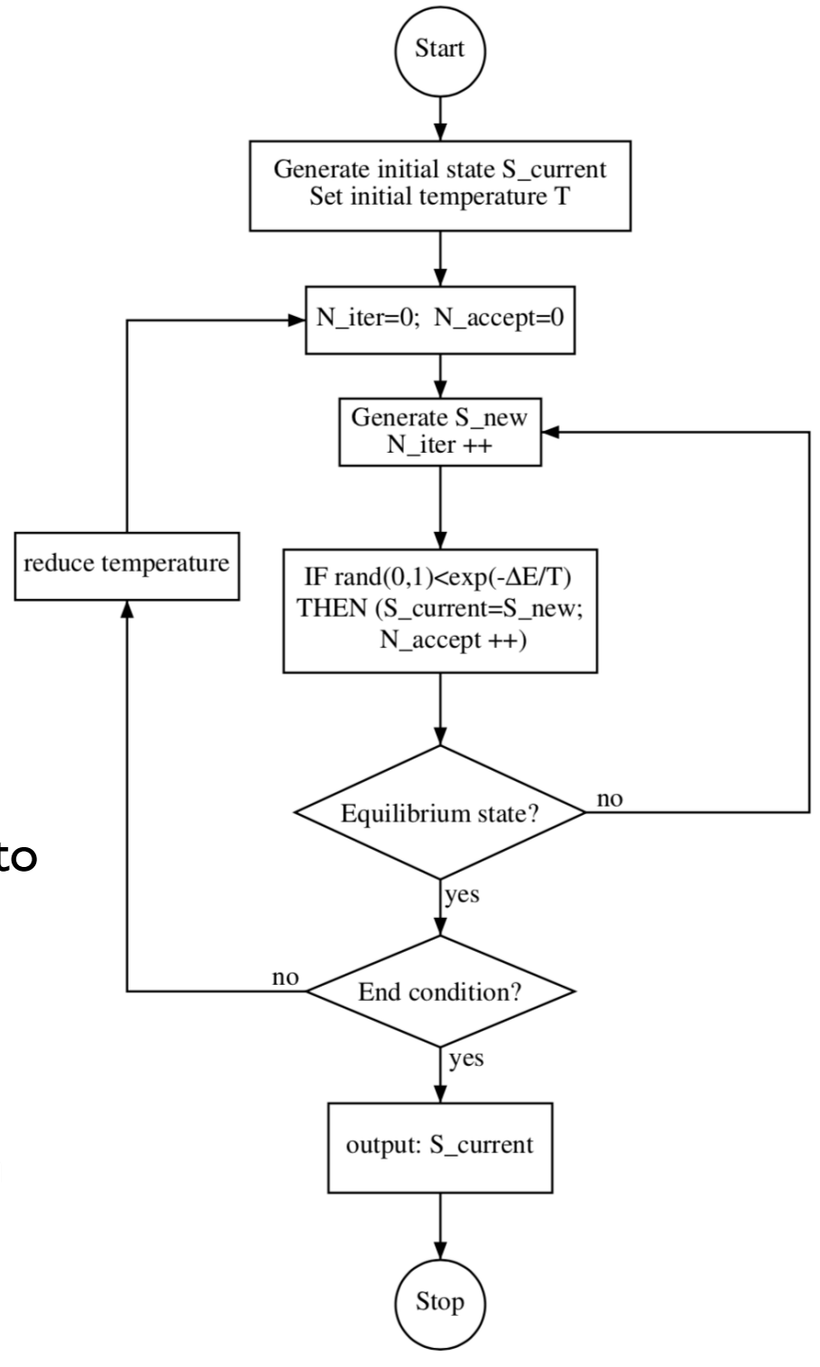
Simulated Annealing

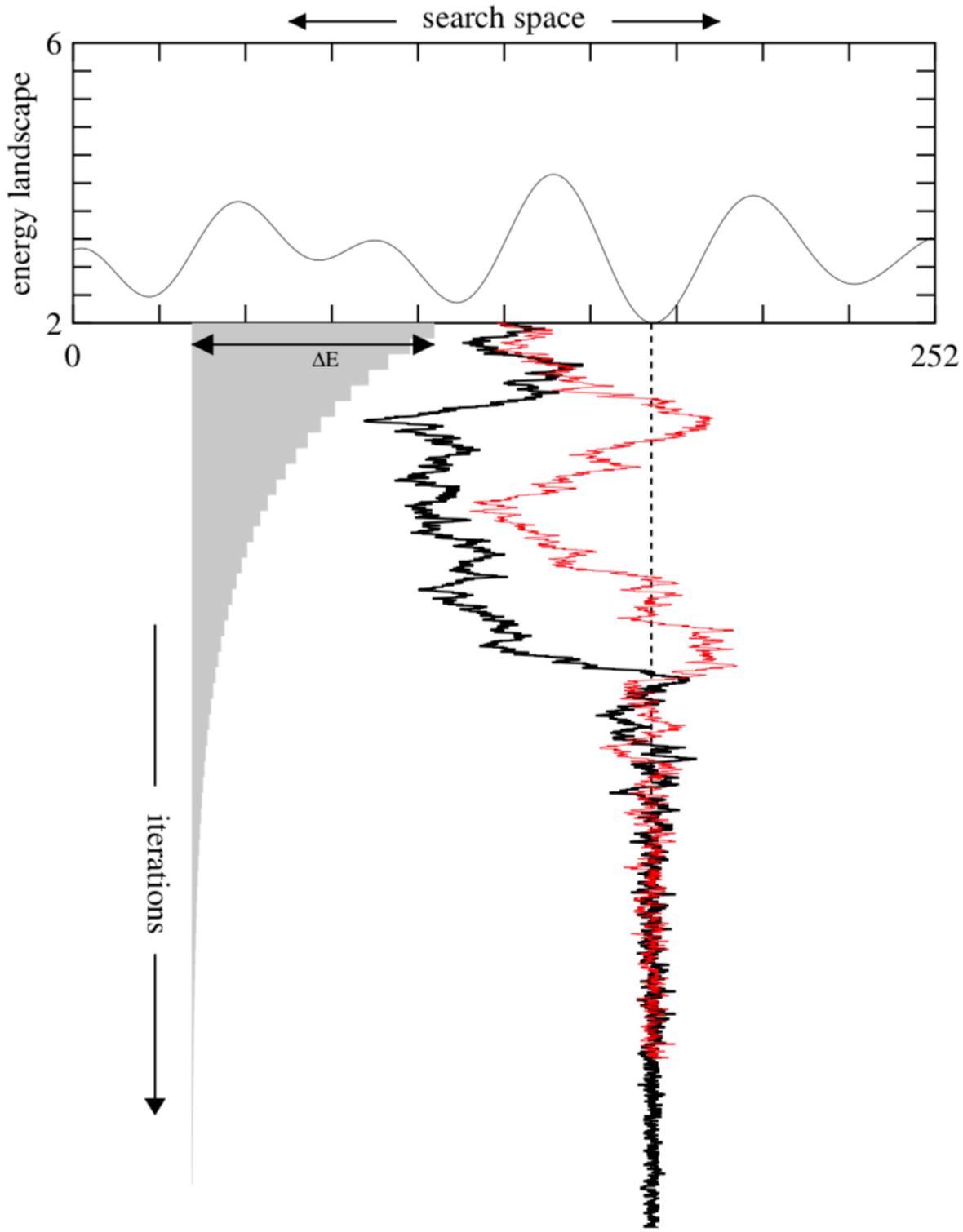
The probability p of accepting the new configuration is defined by the exponential $p = \min(1, e^{(-(\Delta E/T))})$



Simulated Annealing

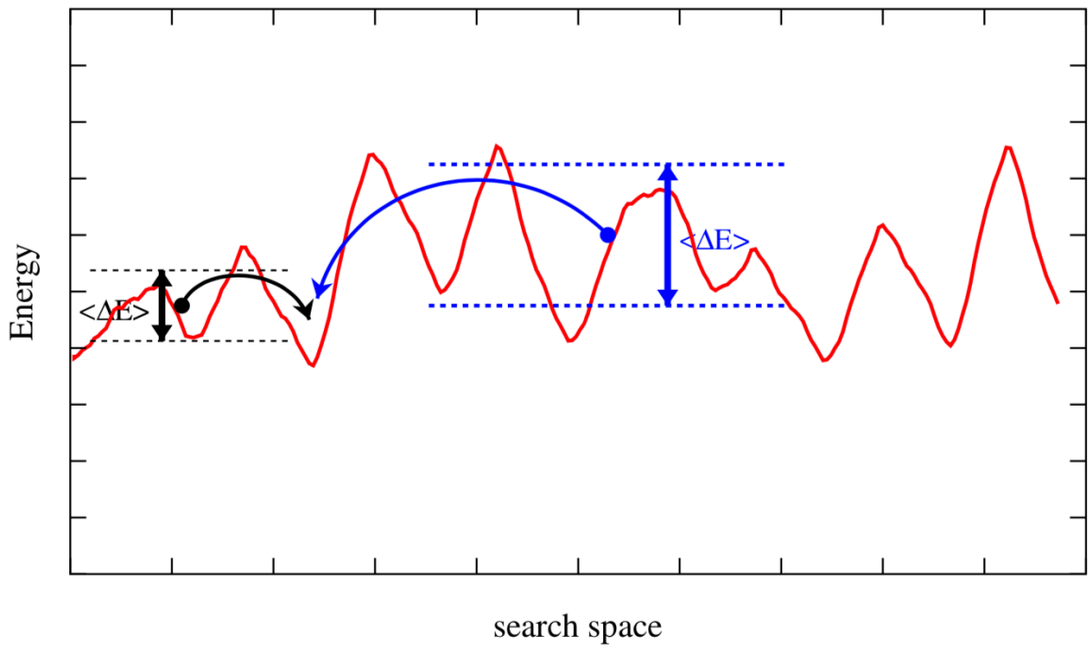
- The temperature is progressively decreased during the search.
- Temperature schedule / cooling schedule
- Often preferred to lower the temperature in stages:
    - First constant temperature for a while
- Until search reaches a stationary value of the energy that fluctuates around a given average value that doesn't change any more
- At this point the temperature is decreased to allow the system to achieve convergence to a lower energy state.
- Finally, after several stages in which the temperature has been decreased, there are no possible fitness improvements; a state is reached that is to be considered the final one, and the algorithm stops.




search space
energy landscape
6
2
0
252
ΔE
iterations

Simulated Annealing

- Heuristic to choose initial temperature:
  - Perform 100 elementary transformations randomly starting from the initial configuration
  - Compute $<\Delta E>$, i.e., the average of the energy variations observed in these 100 moves
  - Choose an initial acceptance probability p_0 for worsening moves according to the assumed quality of the initial solution. Typically, $p_0=0.5$ if the quality is assumed to be average, and $p_0=0.2$ if it is assumed to be good.
  - After that, T_0 can be computed such that: $\exp(-(<\Delta E>/T))=p_0$
    which means that the temperature is high enough to allow the system to traverse energy barriers of size $<\Delta E>$ with probability p_0.



Simulated Annealing

- Temperature stages: Equilibrium state is reached if 12N elementary transformations have been accepted over a total quantity of 100N tried moves.
  (N is the number of degrees of freedom of the problem, i.e., the number of variables that define the solution)

- Temperature schedule: When equilibrium is reached, the system goes to another temperature stage by decreasing T according to a geometric law: $T_{(k+1)} = 0.9\,T_k$
  (where k = is the stage number)

- Termination condition: If during the last three successive temperature stages the energy E didn't improve then the process is halted.

Contents

- Hill climbing
- Tabu search
- Evaluating search algorithms
- Simulated annealing