Week 2: Search and optimisation

Group 5: 30.01.2024

Outline for the hour

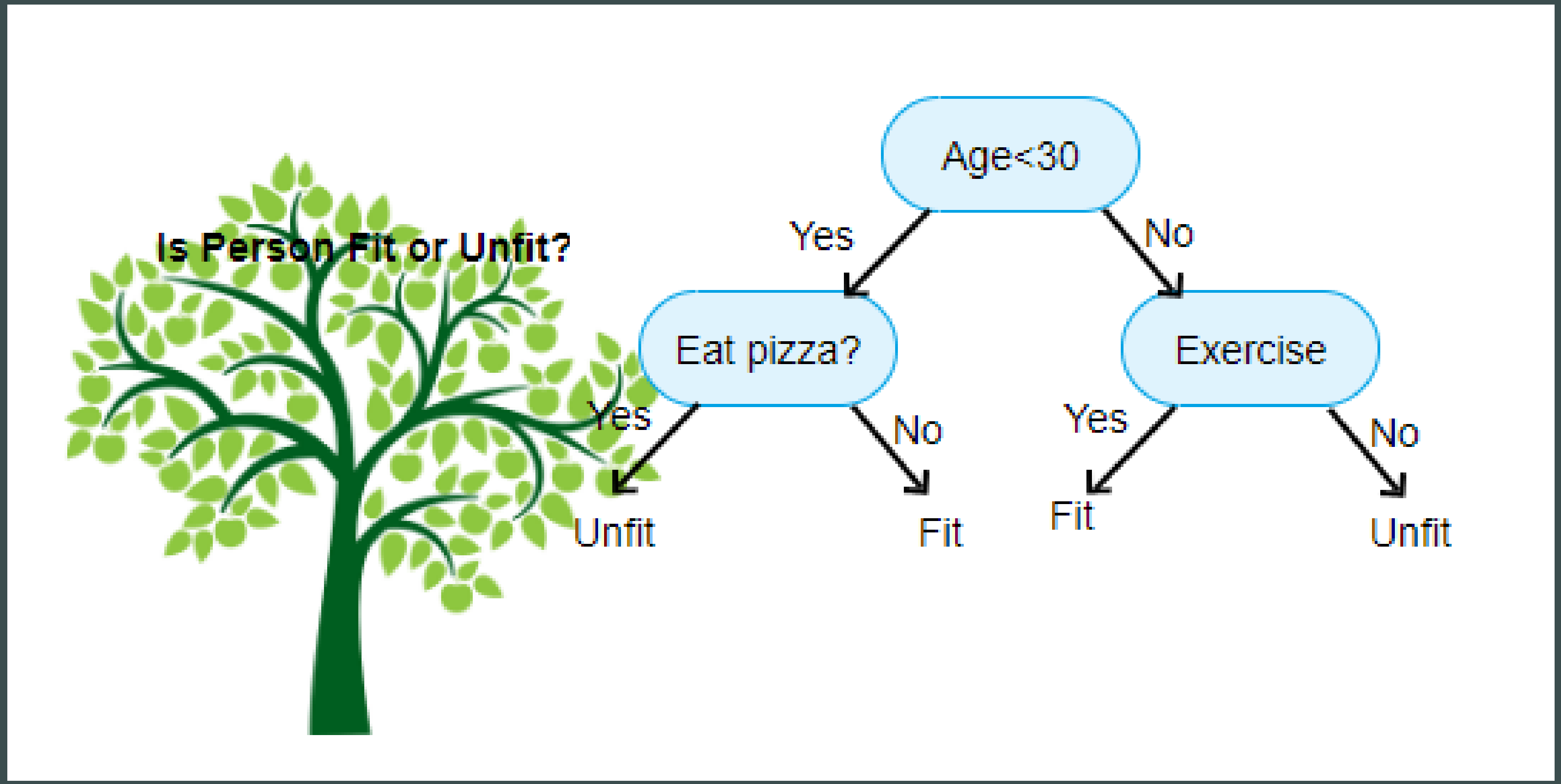
- Trees and random forests
- Exhaustive search
- Greedy search
- Hill climbing
- Simulated annealing
- Gradient descent

Decision trees

- **What is a Decision Tree?**
- A decision tree is a flowchart-like structure where each node represents a decision or a test on an attribute, each branch represents an outcome of that decision, and each leaf node represents the final prediction.

- **How Does it Work?**

1. **Starting Point (Root Node):** The algorithm begins at the root node, which is the first decision or test.
2. **Decision Nodes (Intermediate Nodes):** Each decision node tests a specific attribute, and the outcome determines the next node in the tree. Think of it like answering yes/no questions based on certain features.
3. **Branches:** The branches represent the possible outcomes of a decision, leading to different nodes.
4. **Leaf Nodes:** These are the final outcomes or predictions. They don't have any branches leaving them. Each leaf node corresponds to a specific class or a value, depending on whether it's a classification or regression task.



https://iprathore71.medium.com/complete-guide-to-decision-tree-cee0238128d

- **How it Learns:**

1. **Training:** The algorithm learns from data, figuring out the best questions (features) to ask based on the given examples.
2. **Splitting:** At each decision node, the algorithm chooses the feature that best splits the data into subsets, making the predictions more accurate.
3. **Recursive Process:** The process continues recursively, creating branches and nodes until it reaches a stopping point (leaf nodes).

Advantages:

Easy to understand and interpret.
Requires little data preparation.
Can handle both numerical and categorical data.

Disadvantages:

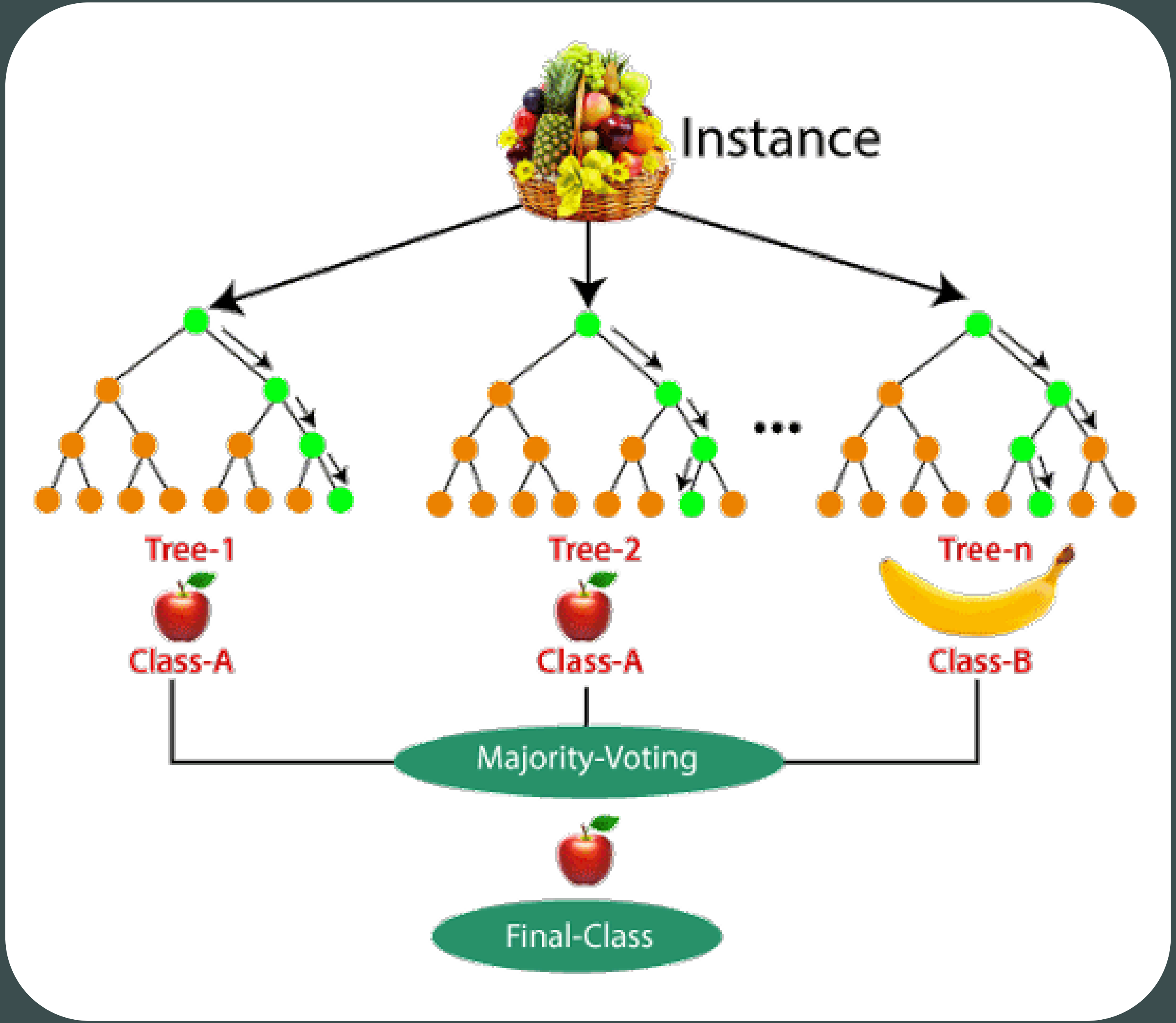
Prone to overfitting (being too specific to the training data).
Can be sensitive to noisy data.
May not perform well on complex relationships in the data.

Random forests

- **What is a Random Forest?**
  A Random Forest is like a group of decision trees working together to make a more robust and accurate prediction.

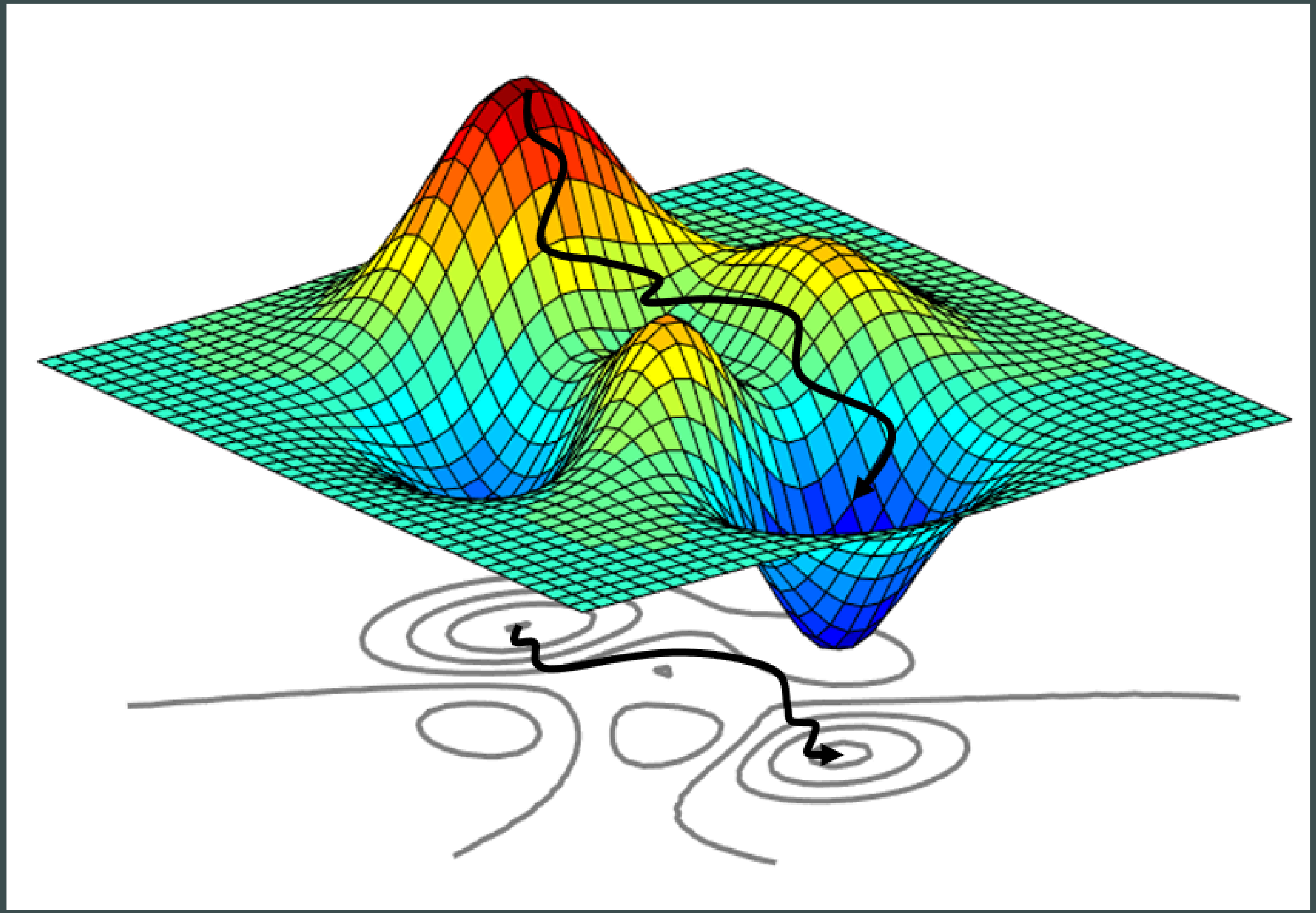
- **How Does it Work?**

- Create Many Decision Trees: Instead of having just one decision tree, a Random Forest creates a bunch of them.
- Random Selection of Features: When building each tree, it doesn't use all the features (questions) available. It randomly selects a subset of features for each tree.
- Training Each Tree: Each decision tree is trained on a random subset of the data, and it makes its own prediction.
- Voting or Averaging: When you want a prediction, each tree "votes" (in classification tasks) or gives a prediction (in regression tasks). The Random Forest combines these votes or predictions.



Optimisation algorithms

- Optimization algorithms are mathematical techniques used to find the best solution to a problem from a set of possible solutions
- optimization algorithms are frequently employed to fine-tune and improve the performance of machine learning models



Exhaustive search

- brute-force search, is a straightforward method that systematically explores all possible solutions to find the optimal one
- Check every single possibility (all **permutations**), always keep track on the best one

- ***Pros*****:** Guarantees finding the best solution, suitable for small problem spaces.
- ***Cons*****:** Inefficient for large search spaces due to the exponential growth of possibilities.

Greedy search

- makes locally optimal choices at each stage with the hope of finding a global optimum.
- ***Process*****:** selects the best available option at each step without reconsidering previous choices.(1 solution at a time, several locally optimal choices in hopes to end up in global optima)

- *Pros*: Simple and computationally efficient for certain problems.
- *Cons*: May not always lead to the globally optimal solution as it lacks a comprehensive view of the entire search space. The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.

Hill climbing

local search algorithm that continually moves towards the direction of increasing elevation in the search space.

- ***Process*:**
- Choose 1 random solution as currently best
- Replace with neighbor solution
- Run n times

  (The algorithm iteratively improves the current solution by making small adjustments)

- *Pros*: Efficient for finding local optima in smooth landscapes.
- *Cons*: Prone to getting stuck in local optima and sensitive to the starting point.

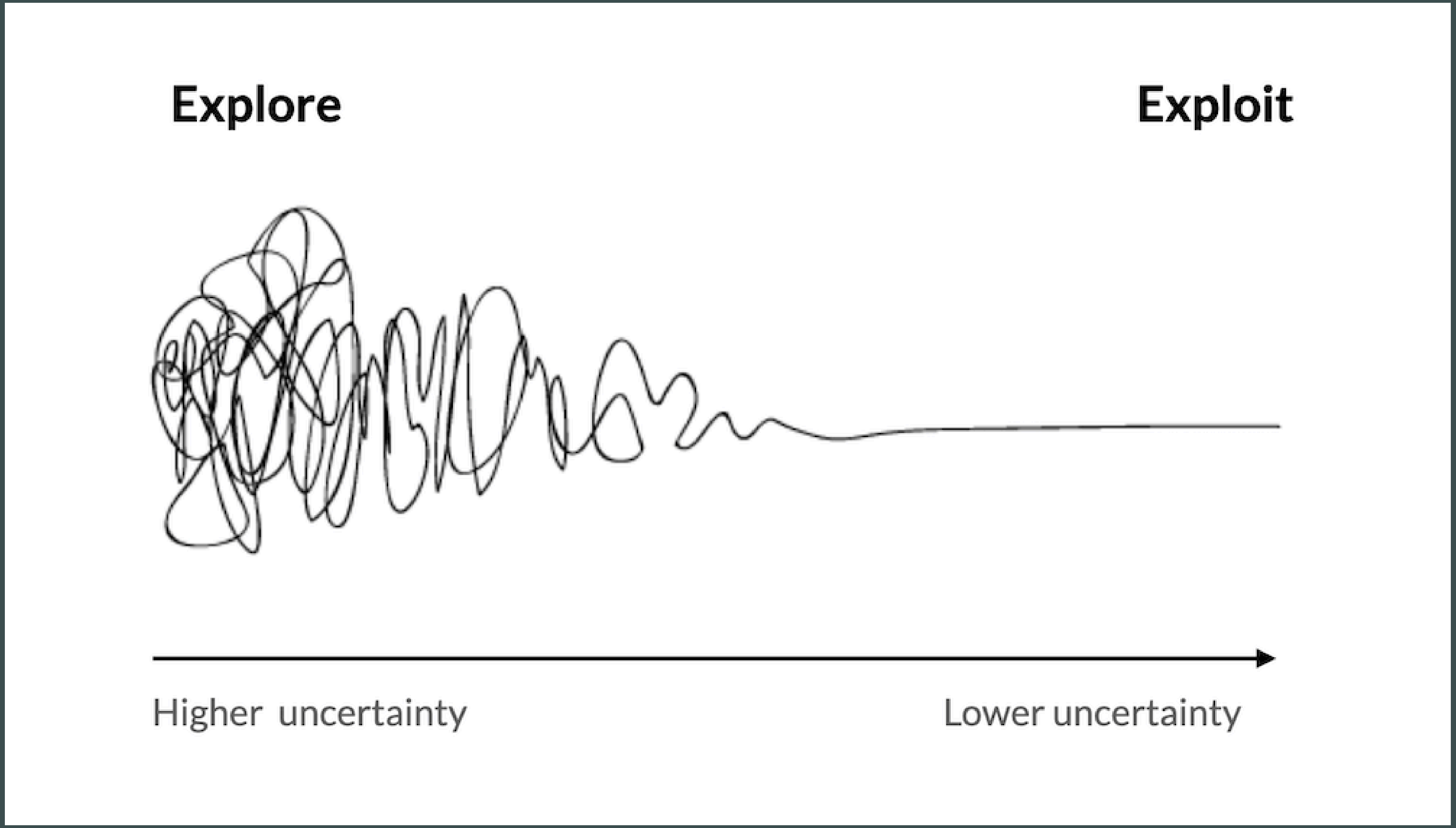
What is the difference between the two?

- **both algorithms are local search strategies**, but the key distinction is in how they navigate the solution space and make decisions.
- Greedy search goes for the best immediate option, while hill climbing moves in the direction of improving the current state

- Hill climbing can backtrack if it reaches a point where no neighboring solutions lead to an improvement.
- Greedy algorithms do not backtrack; once a decision is made, it is final.

Exploration vs exploitation

- Exploration is any action that lets the agent discover new features about the environment, while exploitation is capitalizing on knowledge already gained.



Stochastic vs deterministic

Deterministic Algorithms:

- Deterministic algorithms produce the same output for a given set of inputs and a fixed initial state.
- The behavior of a deterministic algorithm is entirely predictable, and it doesn't involve randomness or chance.

Stochastic Algorithms:

- Stochastic algorithms, on the other hand, involve some element of randomness or chance in their execution.
- The output of a stochastic algorithm can vary on different runs even when given the same inputs and starting conditions.

Simulated annealing

- probabilistic optimization algorithm inspired by the annealing process in metallurgy.

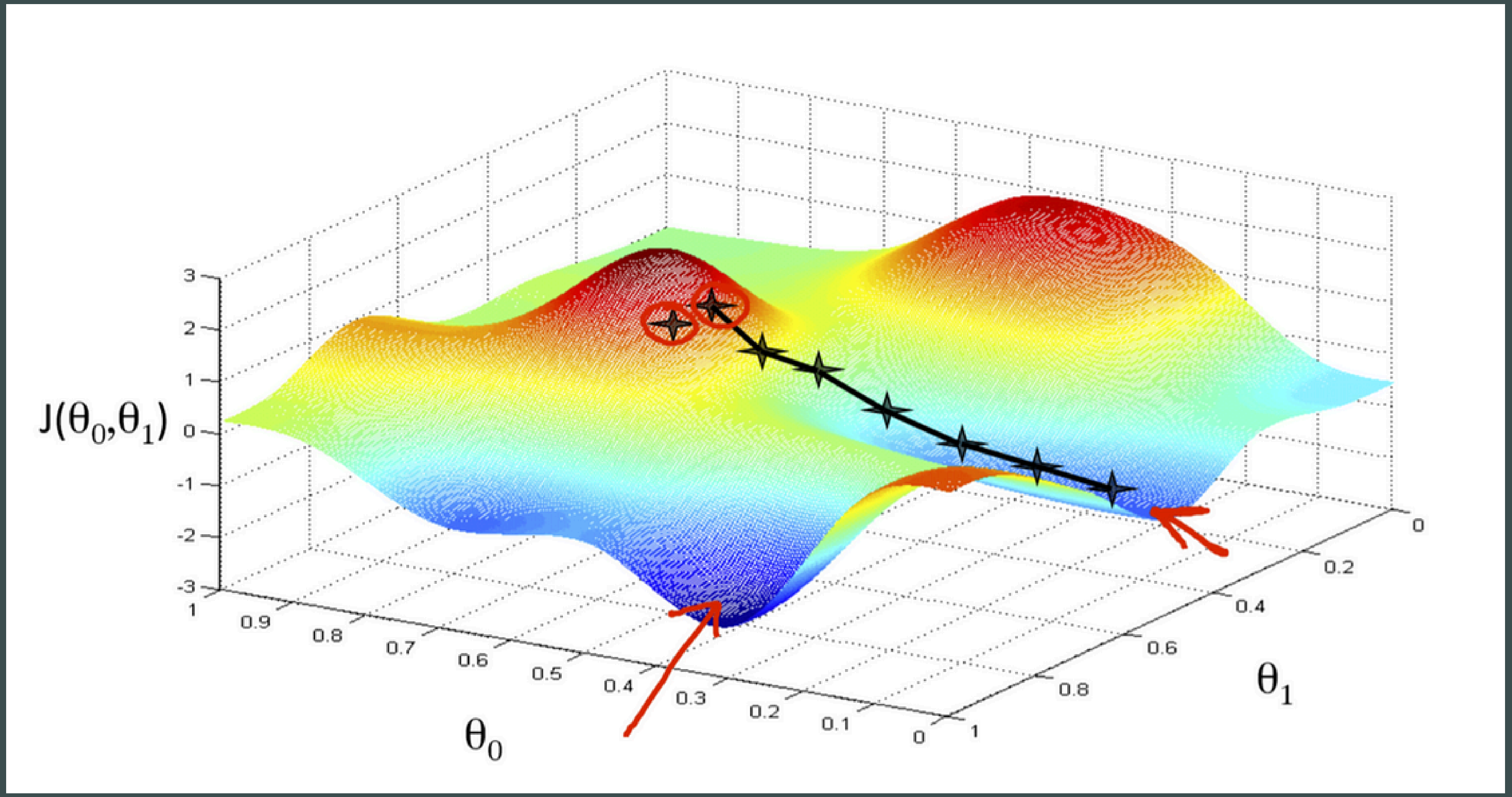
- ***Process*: The algorithm explores the solution space by allowing for occasional "bad" moves with decreasing probability(Temperature) over time.

- ***Pros*: Can escape local optima, suitable for complex and rugged search spaces.
- ***Cons*: Requires careful tuning of parameters and may have slower convergence compared to some deterministic algorithms.

Gradient descent/ascent

- iterative optimization algorithm that minimizes/maximizes a function by **adjusting its parameters in the direction of steepest descent/ascent**.

- ***Process*: ** The algorithm calculates the gradient of the function at each step and updates the parameters accordingly.
- *Pros*: Efficient for convex and smooth optimization problems.
- *Cons*: May converge to a local minimum, sensitivity to initial conditions, and challenges in handling non-convex functions.



Gradient ascent: $x^{(k+1)} = x^{(k)} + \gamma \nabla f(x^{(k)})$

$$x_{n+1} = x_n - \alpha \nabla f(x_n)$$

https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent#:~:text=Gradient%20descent%20minimizes%20differentiable%20functions,direction%20of%20the%20negative%20gradient.

Which of the algorithms is the best?

- **No free lunch theorem!**
  - There is no universal solution for all problems!

- **Exhaustive Search:**

- Small Search Spaces: When the search space is relatively small, and the computational cost of evaluating all possibilities is reasonable.
- Exact Solution Required: In scenarios where an exact solution is crucial, and there is confidence that exhaustive search will yield the best result.
- Testing Solutions: In situations where exhaustive search is used for testing or validating other algorithms by comparing their results to the exhaustive search solution.

- **Greedy Search:**

- Problem with Optimal Substructure: Greedy algorithms are suitable when the problem has optimal substructure, meaning the solution to the overall problem can be constructed from optimal solutions to its subproblems.
- Real-time Applications: In real-time systems where quick decision-making is essential, and a locally optimal solution is acceptable.
- Simplicity: Greedy algorithms are preferred when simplicity is crucial, and the problem can be effectively solved with a series of myopic choices.

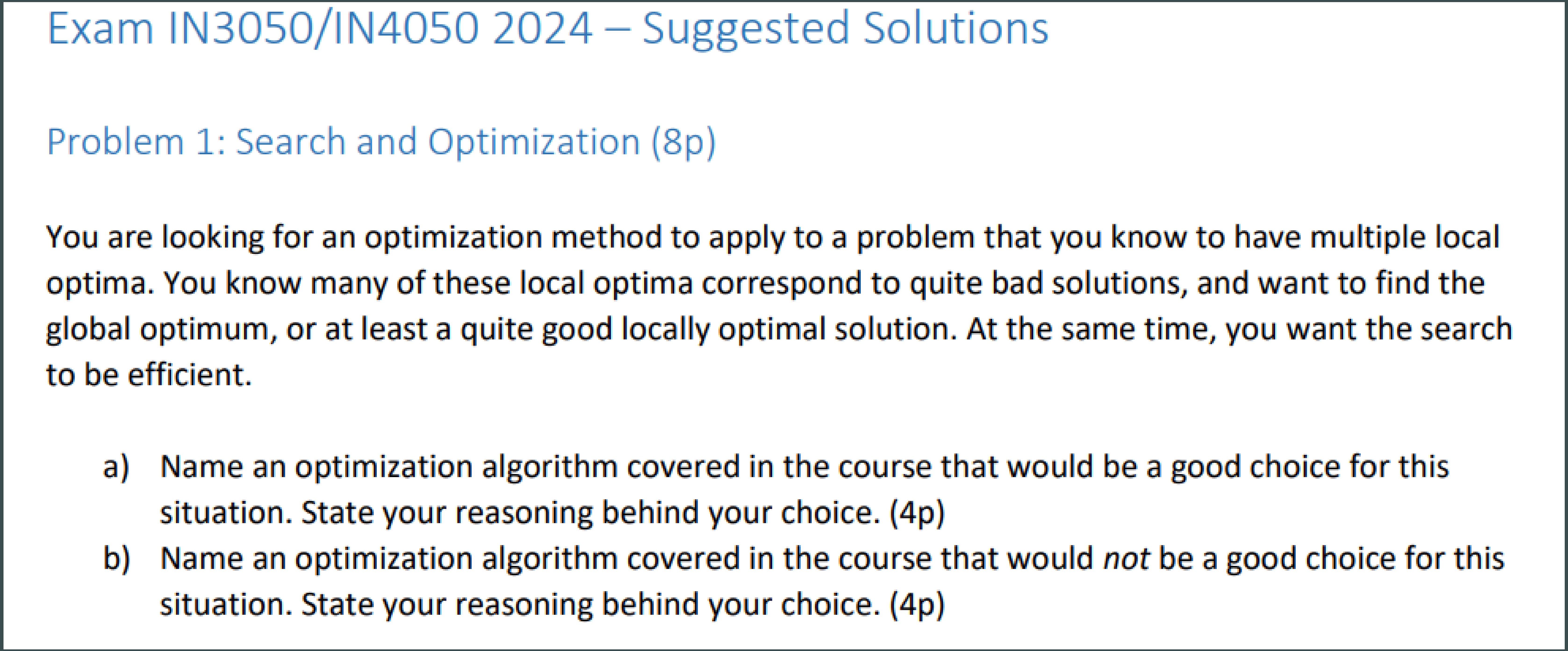
- **Hill Climbing:**
- Local Search Spaces: In scenarios where the solution space is relatively smooth, and local optimization is sufficient to find a good solution.
- Heuristic Improvement: When a good heuristic is available to guide the search towards higher-elevation regions in the solution space.
- Satisficing Solutions: In situations where finding the absolute optimal solution is not necessary, and a satisfactory solution is acceptable.

- **Simulated Annealing:**
- Complex Solution Spaces: Simulated annealing is effective in complex and rugged solution spaces where traditional methods might get stuck in local optima.
- Escape Local Optima: When there is a need to explore a wide range of possibilities and have the ability to escape from local optima to find a better global solution.
- Stochastic Optimization: In scenarios where introducing randomness is beneficial for achieving a more robust and diverse set of solutions.

- **Gradient Descent:**

- Smooth and Continuous Functions: Gradient descent is optimal when dealing with smooth and continuous optimization problems.
- Large Datasets: In machine learning, gradient descent is effective for training models on large datasets due to its efficiency in handling numerous data points.
- Convex Optimization: When dealing with convex optimization problems where a single global minimum exists, gradient descent is an optimal choice.

Typical exam questions



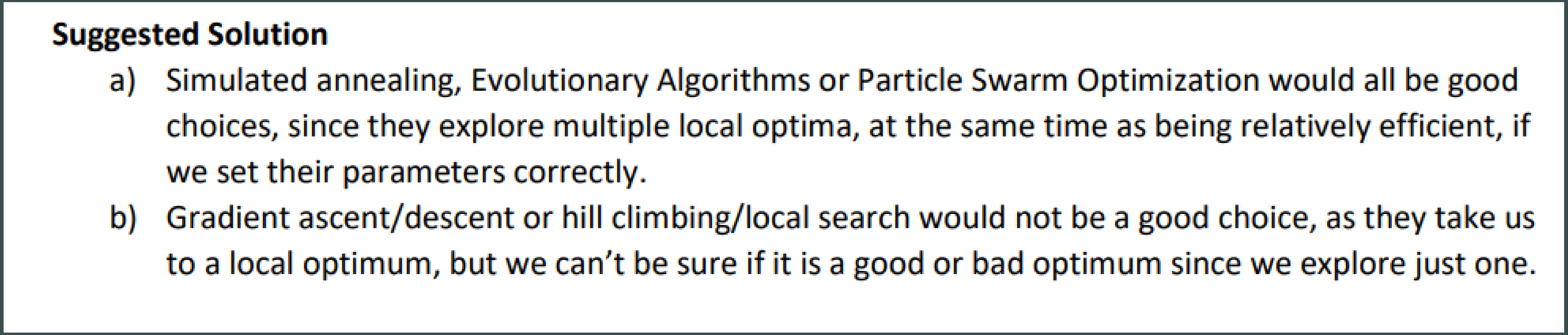
Exam IN3050/IN4050 2024 – Suggested Solutions

Problem 1: Search and Optimization (8p)

You are looking for an optimization method to apply to a problem that you know to have multiple local optima. You know many of these local optima correspond to quite bad solutions, and want to find the global optimum, or at least a quite good locally optimal solution. At the same time, you want the search to be efficient.

a) Name an optimization algorithm covered in the course that would be a good choice for this situation. State your reasoning behind your choice. (4p)
b) Name an optimization algorithm covered in the course that would *not* be a good choice for this situation. State your reasoning behind your choice. (4p)

Typical exam questions



Suggested Solution

a) Simulated annealing, Evolutionary Algorithms or Particle Swarm Optimization would all be good choices, since they explore multiple local optima, at the same time as being relatively efficient, if we set their parameters correctly.

b) Gradient ascent/descent or hill climbing/local search would not be a good choice, as they take us to a local optimum, but we can’t be sure if it is a good or bad optimum since we explore just one.

Weekly exercise:

https://github.uio.no/anastapl/Gruppetimer/blob/main/Week%202%20-%20Optimisation/IN4050%20exercises_w2.ipynb