1 Theoretical

a)

Gradient Descent is an iterative optimization algorithm designed for continuous local search, with the specific goal of finding a global minimum of an objective function. The process commences by selecting a random initial point, x, within the state space. The algorithm's core mechanic is an update rule, $x \leftarrow x - \delta \cdot \nabla f(x)$, which is applied repeatedly. In this formula, $\nabla f(x)$ is the gradient of the function at point x. The gradient, which generalizes the derivative to n dimensions, mathematically points in the direction of the steepest *ascent*. Therefore, by moving in the direction of the negative gradient, the algorithm iteratively steps 'downhill'. The magnitude of each step is governed by the step size parameter δ (also known as the learning rate γ). This procedure is repeated until the gradient's magnitude is approximately zero ($\|\nabla f(x)\| < \epsilon$) which indicates that the search has converged to a stationary point, such as a minimum or saddle point.

However, Gradient Descent possesses several critical limitations. The algorithm's performance is highly sensitive to the choice of the step size parameter. If the step size is set too low, the algorithm will require a very large number of iterations to reach the minimum, resulting in slow convergence. Conversely, if the step size is too high, the updates can be too drastic, causing the algorithm to 'overshoot' the minimum and potentially lead to divergent behavior where it fails to converge at all. A more fundamental limitation arises from the nature of the objective function itself. For complex, non-convex functions, such as those encountered in training neural networks, the algorithm may converge to a **local minimum** rather than the desired **global minimum**. This occurs because a local minimum, like the global minimum, has a gradient of zero, causing the algorithm to stop prematurely. The specific optimum found is, therefore, highly **dependent on the initial starting point** of the search.

Gradient Descent is, however, guaranteed to converge to the global minimum when the objective function is **convex**. Convex optimization problems involve functions that have only a single 'basin', meaning any local minimum found is, by definition, also the global minimum. In this scenario, the primary limitation of getting 'stuck' in a suboptimal local minimum is eliminated.

A clear example of such a function is a simple parabola, like $f(x) = x^2$, which is visualized in the lecture materials. This function is convex and possesses only one minimum, at $x = 0$. Regardless of where the search is initialized (e.g., at $x = -4$ or $x = 4$), the negative gradient will always direct the search 'downhill' toward $x = 0$. As long as an appropriate step size is chosen, Gradient Descent will consistently and successfully converge to this single global minimum.

b)

Based on the lecture materials, Parallel Hill Climbing and Local Beam Search are both advanced local search algorithms designed to address the primary limitation of simple hill climbing: its tendency to become trapped in local optima. Both methods achieve this by maintaining multiple states within the search space simultaneously, rather than the single state used in a basic hill-climbing algorithm.

The fundamental difference and key relationship between them lies in whether these multiple search processes interact.

Parallel Hill Climbing, also referred to as Random-Restart Hill Climbing, operates by conducting multiple, entirely independent hill-climbing searches. The algorithm performs n distinct searches, each originating from a randomly selected initial state. These search processes run in complete isolation and do not communicate. The final solution is determined by simply taking the best-performing state from all n independent trials once they have terminated. This method diversifies the search by exploring different basins of attraction, increasing the probability that one of the runs will land in the basin of the global optimum.

In sharp contrast, **Local Beam Search** employs a cooperative and interdependent strategy. While it also begins with k randomly generated states, the search processes are intrinsically linked. At each iteration, the algorithm generates all successors for all k current states. It then amalgamates these successors into a single pool and selects the k best-valued states from this entire pool to become the new set of current states. This selection mechanism constitutes a significant 'exchange of information' between the k search threads. Unlike parallel hill climbing, where a promising discovery in one search has no effect on the others, Local Beam Search allows the most successful threads to effectively capture the entire search effort. If one state discovers a highly promising region, its successors are likely to dominate the next selection, thereby concentrating the search in that optimal area and discarding the less promising paths found by the other threads.

2 Search algorithm equivalent

a. Local beam search with $k = 1$.

Algorithm Name: Hill-Climbing (specifically, Steepest-Ascent Hill-Climbing)

Explanation: Local beam search maintains a 'beam' of k states at each step. It generates all possible successors for all k states and then selects the k best-valued successors from that combined pool to be the states for the next iteration.

If $k = 1$, the algorithm simplifies as follows:

- It starts with a single ($k = 1$) initial state.
- It generates all successors of that one state.
- From this pool of successors, it selects the 'best k' (which is just the single best successor).
- This single best successor becomes the new current state.

This process of repeatedly moving to the 'highest-valued successor state' after evaluating all neighbors is the exact definition of the Steepest-Ascent Hill-Climbing algorithm.

b. Local beam search with one initial state and no limit on the number of states retained.

Algorithm Name: Breadth-First Search (BFS)

Explanation: This special case sets the initial state $k = 1$ and the retention limit $k = \infty$.

- The algorithm begins with one initial state (the 'root').
- In the first step, it generates all successors of that state (all nodes at depth 1). Because there is 'no limit on the number of states retained', it keeps all of these successors.

- In the next step, it generates all successors for all the states it just retained (all nodes at depth 2) and again keeps all of them.

This process of starting at a single node and systematically exploring the state space layer by layer, exploring all nodes at depth d before any nodes at depth $d+1$, is the definition of Breadth-First Search. The 'best' selection criteria of Local Beam Search becomes irrelevant because retaining an unlimited number of states simply means retaining all of them.

c. Simulated annealing with $T = 0$ at all times (and omitting the termination test).

Algorithm Name: First-Choice Hill-Climbing

Explanation: The core logic of Simulated Annealing is to pick a randomly selected successor of the current state.

- If the move is an improvement ($\Delta E > 0$), it is always accepted.
- If the move is not an improvement ($\Delta E \leq 0$), it is accepted with a probability of $p = e^{\Delta E/T}$.

In the special case where $T = 0$ (or, more precisely, $T \to 0$), we must evaluate this probability. For any non-improving move, ΔE is a negative value (or zero).

- The exponent $\Delta E/T$ will approach $-\infty$.
- The probability of acceptance becomes $\lim_{T\to 0} e^{\Delta E/T} = 0$.

Therefore, the algorithm will never accept a 'down-hill' or 'flat' move. Its logic simplifies to:

1. Select one random neighbor.
2. If it is an improvement ($\Delta E > 0$), move to it.
3. If it is not an improvement ($\Delta E \leq 0$), the probability of moving is 0, so the algorithm stays at the current state and the loop repeats (effectively sampling another neighbor).

This process of generating neighbors one at a time and moving to the first one that provides an improvement matches the description of First-Choice Hill-Climbing.

3 Hill-Climbing Search

To apply the hill climbing approach to the 3X3 grid puzzle, we consider each of the board configurations as a possible state, and use a heuristic (here - number of misplaced tiles) to decide the next state.

$$heuristic(state) = \text{number of misplaced tiles}$$

Step 1. Start by computing the initial state's heuristic value.

Step 2. Generate the next valid neighboring states by sliding the blank tile to the 4 directions (but only if those moves stay inside the board).

Step 3. Compute the heuristic values of all the neighboring states and select the state with the least heuristic value.

Step 4. If the heuristic value of the selected state is less than the heuristic value of the current state, move to that state and go to step 2.

Step 5. Stop when the goal state is reached or when no neighbor reduces the heuristic value.

This algorithm can successfully reach the goal state with 4 moves. If the goal state is altered to the mention configuration, then the algorithm does not reach the goal state and reaches the local optimum with 4 moves. The local optimum state reached is,

1	2	3
4	5	6
7	8	

This state has a heuristic value of 2.

4 Crossword Puzzles

a)

The problem of fitting words into an empty crossword puzzle could be defined as a classical search problem like this:

(a) state space: a crossword grid with any number of word gaps filled

(b) initial state: completely empty crossword grid

(c) actions: choosing an item from a list of words with the right length and possibly characters on specific positions if required

(d) transition model: fill the corresponding gap with the chosen word

(e) goal state: completely filled crossword grid

A fitting search algorithm could be Stochastic Beam Search, because you may need to try to fill the grid from a previous state if no words for a given gap can be found. A heuristic function for this problem could be the number of filled in gaps of the grid.

b)

As a constraint satisfaction problem, it could look like this:

(a) variables: the chosen word for a specific gap

(b) domains: the given list of words which can be chosen. Can be restricted to the possible options of the corresponding gap

(c) constraints: each word has to be the corresponding length and for every overlap of two words, they must have the same character on the position of the overlap

(d) objective: number of satisfied constraints