Here are your exam notes based on the provided documents and exam scope:

Exam Notes: Metaheuristics and Machine Learning

1. Genetic Algorithm (GA)

- **Theoretical Knowledge:**
  - **Concept:** Genetic Algorithms (GAs) are evolutionary algorithms inspired by Darwin's principle of natural selection. They operate on a population of candidate solutions (individuals) to a problem.
  - **Representation:** Individuals are typically encoded as fixed-length linear chromosomes, often binary bit strings, but can also be real numbers or characters. Each gene is a part of the chromosome.
  - **Process:**
    - **Initialization:** An initial population of individuals is usually created randomly. The population size is a user-defined parameter.
    - **Fitness Evaluation:** Each individual's quality (fitness) in solving the problem is assessed using a fitness function. This function is predefined and problem-specific.
    - **Selection:** Individuals are chosen from the current population to act as parents for the next generation, typically favoring fitter individuals. Tournament selection is a common method where a fixed number of individuals are randomly selected, and the fittest among them is chosen.
    - **Genetic Operators:**
      - **Crossover:** Considered the primary genetic operator. Two parent individuals exchange parts of their chromosomes at randomly selected crossover point(s) to create offspring. The likelihood is determined by a crossover probability (pc). Single-point crossover is one type.
      - **Mutation:** Applied to a single individual after crossover, introducing small, random changes to its chromosome to provide diversity and prevent premature convergence to local optima. The mutation rate (pm) is usually lower than the crossover rate. Flip bit mutation is an example.
    - **Population Update/Replacement:** The new generation of offspring replaces the old population. Common methods are generational replacement (entire population replaced) and steady-state replacement (a specific number of individuals replaced).
    - **Termination:** The iterative process (generations) continues until a predefined stopping criterion is met (e.g., maximum number of generations, a near-optimal solution found). The best solution found is then returned.
  - **Encoding Schemes:** Examples include binary encoding (genes are 1s or 0s), real number encoding, and character encoding. The range of possible values for each

gene is specified beforehand.

- **Application Knowledge:**
    - GAs search for a solution in a solution space.
    - Can be used to solve problems like the Traveling Salesman Problem (TSP), where a chromosome represents a tour of cities.

2. Grammatical Evolution (GE)

- **Theoretical Knowledge:**
    - **Concept:** Grammatical Evolution (GE) is an extension of Genetic Programming (GP). It uses variable-length binary string chromosomes.
    - **Representation:** Each gene in the chromosome is an 8-bit binary string called a codon. Codons guide the selection of production rules from a Backus-Naur Form (BNF) grammar.
    - **Mapping Process:** GE uses a user-defined BNF grammar to map the variable-length linear genomes (genotype) to executable programs (phenotype). Domain knowledge is incorporated into the grammar. The mapping uses the rule: Rule = (codon_decimal_value) % (Number_of_production_rules). The derivation process is left-to-right, starting with the leftmost non-terminal. If the end of the codon sequence is reached before the derivation is complete, "wrapping" occurs (looping back to the start of the codons). The mapping is deterministic.
    - **BNF Grammar (G):** Represented as a four-tuple <N, T, P, S>, where N is non-terminals, T is terminals, P is production rules mapping N to T, and S is the start symbol (a member of N).
    - **Process:**
        - **Initialization:** A population of variable-length binary strings is randomly generated. Population size and length limits are user-specified.
        - **Mapping:** Genotypes are mapped to phenotypes using the BNF grammar.
        - **Fitness Evaluation:** The fitness of the phenotype is evaluated by applying it to the problem.
        - **Selection:** Common methods include tournament selection and fitness-proportionate selection.
        - **Crossover:** Single-point crossover is widely used, applied to variable-length genomes, resulting in variable-length offspring. A crossover probability rate determines if it's applied. A random crossover point is chosen within the length of the shorter parent.
        - **Mutation:** Applied similarly to GAs (e.g., bit mutation).
        - **Population Replacement:** Can be generational or steady-state; elitism may also be applied.
        - **Termination:** Occurs after a certain number of generations or if a problem-specific solution is found.
- **Application Knowledge:**
    - GE searches for a program in a program space to solve a given problem.
    - Draws inspiration from molecular biology (DNA genotype to protein phenotype).

3. Genetic Programming (GP)

- **Theoretical Knowledge:**
  - **Concept:** Genetic Programming (GP) is an Evolutionary Algorithm that explores a program space. It's considered an extension of GAs. The aim is to evolve fitter computer programs.
  - **Representation:** Individuals are computer programs, traditionally represented as syntax trees. These trees can be converted to executable expressions (e.g., prefix notation). Each node is a gene.
    - **Internal Nodes:** Functions (operators).
    - **External Nodes (Leaves):** Terminals (inputs to the program).
    - **Function Set:** Application-specific operators (e.g., arithmetic, mathematical, logical, user-defined). Operators have an arity (number of inputs).
    - **Terminal Set:** Inputs for the GP program.
  - **Properties of Function and Terminal Sets:**
    - **Closure Property:** Output from any function/terminal must be valid input for all other functions. Often requires modifying operators (e.g., protected division to handle division by zero).
    - **Sufficiency Property:** Elements must be capable of representing the solution.
  - **Process:**
    - **Initialization:** A population of programs is randomly generated. Tree generation methods include:
      - **Full method:** All nodes up to (max_depth - 1) are functions; nodes at max_depth are terminals.
      - **Grow method:** Creates trees of variable length; nodes between root and (max_depth - 1) can be functions or terminals.
      - **Ramped half-and-half:** Combines full and grow methods. Maximum tree depth (nodes between end node and root) is user-defined.
    - **Fitness Evaluation:** The effectiveness of a program is measured by a fitness function, which is problem-dependent. Programs are applied to a training set (fitness cases). Fitness can be accuracy, speed, etc.
    - **Selection:** Biased towards fitter individuals. Common methods are tournament selection and fitness-proportionate selection.
    - **Genetic Operators:**
      - **Crossover (Subtree Crossover):** Two parent programs exchange subtree branches. A random crossover point is selected on each parent, and the subtrees (crossover fragments) are swapped to create two offspring. Offspring size must not exceed a depth limit (pruning may be needed).
      - **Mutation:** Increases diversity.
        - **Grow Mutation:** Randomly selects a terminal and replaces it

with a subtree, increasing tree size.
            - **Shrink Mutation:** Replaces a randomly selected subtree with a randomly created terminal node. Resultant offspring must conform to specified offspring depth; mutation depth parameter controls subtree size.
        - **Reproduction:** Copying individuals to the next generation.
    - **Population Replacement:** Most frequently generational or steady-state. Steady-state replaces one member based on fitness.
    - **Termination:** Usually a maximum number of generations or when a problem-specific solution is met.
- **Application Knowledge:**
    - GP searches for a program in a program space, unlike GAs which search in a solution space.
    - **Symbolic Regression:** Discovering mathematical expressions/models that fit data. Crucial in data mining, scientific modeling, financial analysis.
    - **Automatic Programming:** Generating programs for tasks like sorting, control systems, game AI.
    - **Machine Learning and Classification:** Evolving classification rules, decision trees.
    - **Control Systems:** Designing controllers for robots, autonomous vehicles.
    - **Circuit Design:** Evolving electronic circuits.
    - **Image and Signal Processing:** Developing algorithms for image recognition, signal filtering.
    - **Financial Modeling:** Creating predictive models for stock trends, risk assessment.
    - **Game Playing:** Evolving strategies for games.
    - **Robotics:** Creating control programs for robots.
    - **Optimization:** Evolving functions to optimize complex systems.
    - Valuable when underlying relationships are complex or unknown.

4. Particle Swarm Optimization (PSO)

- **Theoretical Knowledge:**
    - **Concept:** A population-based optimization technique inspired by the social behavior of bird flocking or fish schooling. It's a metaheuristic that iteratively improves a population of candidate solutions (particles).
    - **Continuous Optimization:** Aims to find optimal values of decision variables within a continuous range.
    - **Particles:** Each particle represents a potential solution and has a position and velocity in the search space. Particles move towards optimal particles based on their own experience and the experience of neighboring particles.
    - **Movement:** Determined by a function of their position and velocity, influenced by local and global best positions.
    - **Fitness:** A fitness value is associated with each particle, calculated by a

problem-dependent fitness function.

- **Process:**
  - **Initialization:** The swarm is initialized with random particles (positions and velocities).
  - **Evaluation:** Fitness of each particle is evaluated using the objective function.
  - **Update Personal Best (pbest):** If a particle's current fitness is better than its personal best fitness, its pbest position and fitness are updated.
  - **Update Global Best (gbest):** If a particle's current fitness is better than the global best fitness found by the swarm, the gbest position and fitness are updated.
  - **Velocity Update:** Each particle's velocity is updated using the equation: $vi(t+1)=w \cdot vi(t)+c1 \cdot r1 \cdot (pi-xi(t))+c2 \cdot r2 \cdot (g-xi(t))$ Where:
    - $vi(t)$: current velocity of particle i at time t.
    - $vi(t+1)$: updated velocity.
    - w: inertia weight (balances exploration and exploitation).
    - c1: cognitive acceleration coefficient (influence of particle's pbest).
    - c2: social acceleration coefficient (influence of swarm's gbest).
    - r1,r2: random numbers between 0 and 1 (stochasticity).
    - pi: personal best position of particle i.
    - $xi(t)$: current position of particle i.
    - g: global best position found by the swarm. The velocity update has three components: Inertia, Cognitive, and Social.
  - **Position Update:** Each particle's position is updated using: $xi(t+1)=xi(t)+vi(t+1)$.
  - **Termination:** Continues until a termination condition is met (e.g., max iterations, time limit, min improvement). The global best position and fitness are returned.
- **Parameters:**
  - **Swarm Size:** Number of particles.
  - **Maximum Velocity:** Limits velocity change to prevent overshooting.
  - **Inertia Weight (w):** Balances exploration and exploitation.
  - **Acceleration Coefficients (c1,c2):** Influence of personal and global bests.
  - **Neighborhood Topology:** How particles communicate (global, local, hybrid).
  - **Local Search Strategy:** Can be combined with local search.
  - **Termination Criteria.**
- **Variations:**
  - **Constriction Factor PSO:** Introduces a constriction factor to limit max velocity change, increasing convergence.
  - **Inertia Weight PSO:** Modifies velocity update with an inertia weight term.
  - **Cooperative PSO:** Multiple sub-swarms work independently and exchange information.

        - **Hybrid PSO:** Combines PSO with other algorithms (e.g., GAs, simulated annealing).
        - **Multi-objective PSO:** Extends PSO for problems with multiple objectives using Pareto optimization.
- **Application Knowledge:**
    - Used in continuous optimization problems.
    - Representation involves particles with velocity and direction.
    - Search space can be solution or program space.

5. Ant Colony Optimization (ACO)

- **Theoretical Knowledge:**
    - **Concept:** A swarm intelligence algorithm and metaheuristic that mimics the foraging behavior of ants to solve optimization problems. It's a multipoint search technique using a population (colony) of artificial ants.
    - **Natural Inspiration:** Ants find the shortest path to a food source by depositing pheromones on the ground, creating trails. Stronger trails attract more ants; pheromones evaporate over time unless reinforced.
    - **Artificial Ants:** Simulate this process. At each decision point, ants choose a solution component. Solutions are constructed by traversing a graph where vertices/edges represent solution components. Each trail has a pheromone parameter (τ_{ij}) and a heuristic value (η_{ij}).
    - **Process:**
        - **Initialization:** Set initial pheromone values (τ_{ij}) and parameters. Parameters include the number of ants.
        - **Solution Construction:** Each ant starts at a random/different node and incrementally builds a solution by moving from one vertex/component to the next. The choice of the next component C_{ij} is based on pheromone and heuristic information using a probability rule.
            - **Probability Rule (example):** $p_{ij}(t)=\Sigma_{k\in N_i}[\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}$ Where:
                - $\tau_{ij}(t)$: pheromone on edge (i,j) at time t.
                - η_{ij}: heuristic value (e.g., inverse of distance).
                - α: parameter controlling pheromone influence.
                - β: parameter controlling heuristic influence.
                - N_i: set of feasible nodes from node i.
        - **Local Search (Optional):** Daemon actions can be applied to complete solutions.
        - **Pheromone Update:**
            - Goal: Increase pheromones on high-quality trails, decrease on low-quality.
            - Evaporation: Pheromones on all trails are reduced by a pheromone evaporation rate (ρ).
            - Deposition: Pheromone is added to trails used by ants based on

solution quality.
            - **Update Rule (example):** $\tau_{ij}(t+1)=(1-\rho)\cdot\tau_{ij}(t)+\Delta\tau_{ij}$ Where:
                - ρ: pheromone evaporation rate ($0<\rho<1$).
                - $\Delta\tau_{ij}=\sum_{k=1}^{m}\Delta\tau_{ij}^{k}$ (sum of pheromone deposited by m ants).
                - $\Delta\tau_{ij}^{k}=Q/L_k$ if ant k used edge (i,j), else 0 (where Q is a constant, L_k is solution length/quality for ant k).
        - **Termination:** Continues until termination criterion is met (e.g., number of iterations, CPU time).
    - **Parameters:**
        - **Number of Ants.**
        - **Pheromone Evaporation Rate (ρ).**
        - **Pheromone Intensity (Q).**
        - **Heuristic Information (η_{ij}).**
        - **Ant Decision Rule (parameters α,β).**
        - **Local Search Strategy.**
        - **Termination Criteria.**
    - **Variations:** Differ in probability rules and pheromone update rules.
        - **Max-Min Ant System (MMAS):** Limits pheromone deposition to prevent premature convergence.
        - **Ant System (AS):** Original version, simple update rule.
        - **Rank-Based Ant System (RAS):** Rank-based selection for ant movement.
        - **Ant Colony System (ACS):** Multiple colonies, global pheromone update.
        - **Elitist Ant System (EAS):** Favors best ants in pheromone update.
- **Application Knowledge:**
    - Representation uses artificial ants with pheromone and heuristic values.
    - Search space can be solution or program space.
    - **Routing and Transportation:** Vehicle routing, supply chain management.
    - **Telecommunications:** Routing in sensor networks, resource allocation.
    - **Manufacturing and Production:** Scheduling, layout design, quality control.
    - **Bioinformatics:** Protein folding, sequence alignment.
    - **Financial Planning:** Portfolio optimization, risk management.
    - Versatile for finding near-optimal solutions and adapting to changing environments.

6. K-Nearest Neighbor (KNN)

- **Theoretical Knowledge:**
    - **Concept:** A straightforward machine learning method for classification or regression. It's a non-parametric, instance-based learning algorithm (memorizes training data).
    - **Principle:** Assigns output to a new instance based on the output of its 'K' closest neighbors in the training data.
    - **Process:**
        - **Data Preparation:** Normalize or scale features if necessary to ensure equal

        contribution to distance calculation.
        - **Choose K:** Select the number of neighbors (K) to consider.
        - **Calculate Distances:** For a new data point, calculate its distance to all points in the training dataset using a chosen distance metric (e.g., Euclidean, Manhattan).
            - Euclidean distance: $d(Q,P_i)=\Sigma(x_{Qj}-x_{Pij})2$
        - **Find K-Nearest Neighbors:** Identify the K training points closest to the new data point.
        - **Make a Prediction (Majority Voting for Classification):**
            - **Classification:** Assign the majority class among the K-nearest neighbors to the new data point.
            - **Regression:** Compute the average (or weighted average) of the target values of the K-nearest neighbors.
    - **Characteristics:**
        - Easy to understand and implement.
        - Can be resource-intensive with large datasets.
        - Vulnerable to noisy or poorly scaled data.
        - The choice of distance metric is important.
- **Application Knowledge:**
    - Used in various fields, including recommendation systems.
    - **Example:** Given Class A points: (2,4), (5,8), (1,6) and Class B points: (7,2), (9,5). To classify Q(6,4) with K=3:
        1. Calculate Euclidean distances from Q to all points.
            - d(Q, P1(2,4)) = 4.0 (Class A)
            - d(Q, P2(5,8)) ≈ 4.12 (Class A)
            - d(Q, P3(1,6)) ≈ 5.39 (Class A)
            - d(Q, P4(7,2)) ≈ 2.24 (Class B)
            - d(Q, P5(9,5)) ≈ 3.16 (Class B)
        2. Identify K=3 nearest neighbors: P4 (B, 2.24), P5 (B, 3.16), P1 (A, 4.0).
        3. Majority class among neighbors: Class B (2 out of 3).
        4. Therefore, Q(6,4) is classified as Class B.

7. Hopfield Neural Network

- **Theoretical Knowledge:**
    - **Concept:** A type of recurrent neural network used for pattern association, specifically autoassociative memory. It's a single-layer neural network.
    - **Pattern Association:** The network memorizes patterns and can recognize them later, even if the input is noisy or incomplete (containing missing or incorrect components).
    - **Autoassociative Memory:** The input and output vectors are the same (the network learns to recall the input pattern itself).
    - **Recurrent:** Contains feedback connections, allowing information from prior inputs/states to influence current processing.

- **Representation:**
    - Input/output vectors can be binary or bipolar.
    - The network is trained by determining a weight matrix (W). If the input vector length is n, the weight matrix is n×n.
- **Training (Determining Weight Matrix):**
    - Weights on the diagonal (w_{ii}) are 0.
    - **Binary Training Data:** Each weight w_{ij} (for i偃=j) is calculated using: $w_{ij}=\Sigma_e[2p_i(e)-1][2p_j(e)-1]$ (where e is an element/pattern in the training set). This involves converting binary patterns to bipolar (-1, 1) before taking the outer product.
    - **Bipolar Training Data:** Each weight w_{ij} (for i偃=j) is calculated using: $w_{ij}=\Sigma_e p_i(e)p_j(e)$. This is the outer product of the input vectors.
- **Application/Recall Process (Algorithm 6):**
    1. Initialize the output vector y to be the input (potentially corrupted) pattern p.
    2. Iteratively update components of y until convergence (no change in y):
        - Randomly select a component y_i to update.
        - Calculate the net input to unit i: $s_i=p_i+\Sigma_{j=1}^{n}y_jw_{ji}$ (Note: the provided text uses p_i here, but typically in recall, the original pattern p is not used directly in the sum after initialization; the current state y is used. The lecture slide example uses the current y values for the sum). The formula in the notes (Eq for s_i) refers to the i-th component of the input vector p_i plus the weighted sum of the *output vector* y and the i-th column of W. For an autoassociative network, p is the initial (possibly corrupted) state of y.
        - Apply activation function:
            - If $s_i>\theta_i$, then $y_i=1$ (for binary).
            - If $s_i<\theta_i$, then $y_i=0$ (for binary) or -1 (for bipolar).
            - If $s_i=\theta_i$, then $y_i=y_i$ (no change). θ_i is a threshold for each output component, often set to 0.
        - The input vector p is updated according to changes in y (in the sense that y becomes the new state for the next component update).
- **Convergence:** The algorithm converges when there is no change in the output vector y over an epoch.

- **Application Knowledge:**
    - Used to repair input vectors with missing or incorrect components.
    - **Example:** Train to store pattern [1 1 1 0] (binary), $\theta_i=0$.
        1. Convert to bipolar for weight calculation: [1 1 1 -1].
        2. Calculate Weight Matrix W (outer product, diagonals = 0): $W=011-1101-1110-1-1-1-10$
        3. Correct a new pattern, e.g., [0 0 1 0].
            - Initial: y=p=[0 0 1 0].
            - Update y_1:

$s1=p1+(y1w11+y2w21+y3w31+y4w41)=0+(0\cdot 0+0\cdot 1+1\cdot 1+0\cdot -1)=1$. Since $s1>0$, $y1=1$. So $y=[1\ 0\ 1\ 0]$. (Note: The calculation in the lecture slide uses $s1$=sum of products of current y and corresponding column of W, which is $0\cdot 0+0\cdot 1+1\cdot 1+0\cdot(-1)=1$. The text adds $p1$, which is 0, so the result is the same here).

- Update y4 (randomly chosen): $s4=p4+(y1w14+y2w24+y3w34+y4w44)=0+(1\cdot -1+0\cdot -1+1\cdot -1+0\cdot 0)=-2$. Since $s4<0$, $y4=0$. So $y=[1\ 0\ 1\ 0]$.
- Update y2: $s2=p2+(y1w12+y2w22+y3w32+y4w42)=0+(1\cdot 1+0\cdot 0+1\cdot 1+0\cdot -1)=2$. Since $s2>0$, $y2=1$. So $y=[1\ 1\ 1\ 0]$.
- Update y3: $s3=p3+(y1w13+y2w23+y3w33+y4w43)=1+(1\cdot 1+1\cdot 1+1\cdot 0+0\cdot -1)=1+2=3$. Since $s3>0$, $y3=1$. So $y=[1\ 1\ 1\ 0]$. The network corrected the pattern to the stored pattern $[1\ 1\ 1\ 0]$. Further epochs would confirm convergence.