# ACO-Based Text Summarization System

## 1. Project Title

Ant Colony Optimization for Extractive Text Summarization

## 2. Objective

The aim of this project is to develop an intelligent system capable of generating meaningful and concise summaries from longer text documents using the Ant Colony Optimization (ACO) algorithm. This system mimics the collective behavior of ants in nature to explore paths and reinforce better solutions over time. In the context of summarization, this translates to selecting a set of sentences that best represent the content of an article while maintaining diversity and brevity. This approach is designed to be both flexible and robust, allowing customization based on sentence count or word limits.

## 3. Methodology

3.1 Preprocessing:  
To ensure accurate summarization, the text undergoes preprocessing steps such as removing stopwords, punctuation, and stemming. These steps reduce noise and normalize the text. Sentences are tokenized using NLTK to preserve structure while simplifying content.  
  
3.2 Similarity Matrix Construction:  
Each sentence is vectorized using TF-IDF to reflect its relative importance. Then, a similarity matrix is built using cosine similarity to measure how closely related each sentence is to the others. This matrix serves as the heuristic input for ACO.

## 4. Ant Colony Optimization

ACO is a probabilistic technique that models the way ants find optimal paths in nature. Each 'ant' in the algorithm represents a virtual agent that constructs a solution (a set of sentences) by probabilistically selecting the next sentence based on two factors:  
- Pheromone: a trail value that reinforces good solutions.  
- Heuristic: similarity scores derived from the TF-IDF matrix.  
  
The process includes:  
- Initialization: Small positive pheromone values and similarity-based heuristics are initialized.  
- Solution Construction: Each ant builds a candidate summary by selecting sentences not previously visited.  
- Evaluation: Each solution is scored based on its coverage (how well it represents the document) and diversity (to avoid redundancy).  
- Pheromone Update: Good solutions are rewarded by reinforcing the pheromone trail between selected sentence pairs.  
  
Parameters used include:  
- Number of Ants: 20  
- Iterations: 100  
- Alpha and Beta: Weighting factors for pheromone and heuristic influence respectively.  
- Evaporation Rate: Controls how quickly past knowledge fades.  
- Summary Size: Controls the number of sentences in the output.  
- Target Word Count: Applies a penalty if exceeded, to control verbosity.

## 5. Summary Generation

After the best-performing solution is identified, the selected sentence indices are sorted in their original order to preserve coherence. The sentences are concatenated into a human-readable paragraph, which forms the final summary. This ensures the output is both algorithmically optimal and linguistically natural.

## 6. Code Structure

The program is modular and cleanly organized, consisting of the following main components:  
  
- preprocess\_text(): Cleans and tokenizes the input text.  
- create\_similarity\_matrix(): Computes cosine similarity from TF-IDF vectors.  
- ACOForSummarization class: Encapsulates the entire ACO logic, including pheromone updates and solution evaluation.  
- generate\_summary(): Ties all modules together to produce the final summary.  
- process\_csv(): Applies summarization to each row of a dataset and outputs structured results.

## 7. Output

For each article, the system generates:  
- A 3-sentence summary  
- A 2-sentence summary  
- A word-limited summary (e.g., 40 words)  
Additionally, it calculates and compares word counts with human-written highlights for evaluation. All results are stored in a CSV file for future analysis or visualization.

## 8. Strengths

The ACO-based summarizer stands out in several areas:  
- It intelligently balances sentence importance and diversity, reducing redundancy.  
- It’s adaptable: users can configure sentence count or word count.  
- It’s robust and numerically stable thanks to clipping and fallback logic.  
- It’s interpretable and customizable for research or production environments.

## 9. Potential Improvements

Future improvements could include:  
- Integrating evaluation metrics like ROUGE or BLEU to assess summary quality against human references.  
- Leveraging multiprocessing to parallelize ACO iterations for speed.  
- Incorporating abstractive models like BART or T5 for hybrid summarization.  
- Building a web-based GUI or RESTful API for user interaction.

## 10. Sample Output (Illustrative)

Original Word Count: 358  
3-Sentence Summary: [Selected sentences appear here...]  
Summary Word Count: 67  
Highlights Word Count: 71  
This illustrates how the algorithm condenses content while maintaining fidelity.