CODE BOOKLET

COMPARATIVE STUDY OF BIO-INSPIRED OPTIMIZATION ALGORITHMS FOR FEATURE SELECTION IN INTRUSION DETECTION SYSTEM

ACADEMIC SESSION: 2024/2025

STUDENT: ABDUL AZIM BIN ANUAR VEERA (A21EC0001)

PROGRAM: Bachelor of Computer Science (Computer Networks & Security)

SUPERVISOR: ASSOC. PROF. DR. ANAZIDA BINTI ZAINAL

CO-SUPERVISOR:

SUMMARY:

This document holds all the coding for the final year project titled Comparative Study of Bio-Inspired Optimization Algorithm for Feature Selection in Intrusion Detection System. The study examines two bio-inspired algorithms: Bee Colony (BC) and Fish Swarm (FS) in the context of its application to choosing relevant features in an Intrusion Detection System (IDS). On the UNSW-NB15 dataset, the research implements BC and FS to decrease the dimensionality of the data and finally, combines chosen features to a Random Forest-based IDS. It assesses and compares them on the basis of accuracy, F1-score, detection rate etc. The findings indicate that both algorithms can enhance the detection, where BC is a little bit more accurate and FS provides more diversity in features.

Table of Contents

Project Hardware and Software List

HARDWARE	VERSION	EXPLANATION OF USAGE
LAPTOP	ASUS TUF GAMING 15 Intel core i7	Used to perform the entire research including dataset preprocessing, model training and evaluation of results.

SOFTWARE	VERSION	EXPLANATION OF USAGE
Google Colab	3.11	Used as main code editor to apply, write, debug code and run experiments.
Microsoft Excel	Excel 2021 MSO	Used to store, organize, and manage dataset for the research
Python	3.11	Programming language used for both algorithms from preprocessing to result evaluation.
Pandas	1.5.3	Used for data manipulation and analysis
Numpy	1.24.0	Used for numerical computing with multi-dimensional arrays
Matpolib	3.6.3	Used for plotting and data visualization
Seaborn	0.11.2	Used for statistical data visualization

Module List and Explanation

Bee Colony Algorithm

MODULE	FUNCTION	USERS
run_bee_colony()	Implements the Bee Colony Algorithm for feature selection. Returns selected features, indices, convergence data, and iteration count.	Feature Selection
evaluate()	(Nested inside run_bee_colony) Evaluates a feature mask using RandomForest and cross-validation. Caches result for efficiency.	Model Training
train_and_evaluate()	Trains a RandomForest model on the selected features and evaluates it on the training set.	Model Training
final_evaluate()	Evaluates the trained model on the test dataset and returns key performance metrics.	Data Evaluation
plot_convergence()	Plots the convergence curve of accuracy over Bee Colony iterations.	Result Visualization
plot_conf_matrix()	Displays the confusion matrix for the predicted vs. actual test labels.	Result Visualization
plot_roc_curve()	Plots the ROC curve of model predictions.	Result Visualization

Fish Swarm Algorithm

MODULE	FUNCTION	USERS
run_fish_swarm()	Runs the Fish Swarm algorithm to pick the best features that help the model work better.	Feature Selection
evaluate()	Tests how well a group of selected features helps the model perform.	Model Training
get_neighbors()	Pick a few other fish (feature sets) for a fish to compare itself with.	Data Selection
move_towards()	Slightly changes a fish's feature set to look more like a better-performing neighbor.	Data Selection
train_and_evaluate()	Trains a Random Forest model on the selected features and prints results on the training data.	Model Training
final_evaluate()	Evaluates the model's performance on the test dataset (real-world data).	Data Evaluation
plot_convergence()	Plots the convergence curve of accuracy over Bee Colony iterations.	Result Visualization
plot_conf_matrix()	Displays the confusion matrix for the predicted vs. actual test labels.	Result Visualization
plot_roc_curve()	Plots the ROC curve of model predictions.	Result Visualization

Code Bee Colony Algorithm

```
# ================================
# SECTION 1: Setup and Imports
# ================================
from google.colab import drive
drive.mount('/content/drive')

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay
# ================================
# SECTION 2: Load and Preprocess Data
# ================================
base_path = '/content/drive/MyDrive/IDS_BioInspired_Colab_Final'
data_path = os.path.join(base_path, 'data')

train_df = pd.read_csv(os.path.join(data_path, 'UNSW_NB15_training-set.csv'))
test_df = pd.read_csv(os.path.join(data_path, 'UNSW_NB15_testing-set.csv'))

drop_cols = ['label', 'attack_cat']
```

```
X_train_raw = train_df.drop(columns=drop_cols)
X_test_raw = test_df.drop(columns=drop_cols)
y_train = train_df['label']
y_test = test_df['label']

numeric_cols = X_train_raw.select_dtypes(include=[np.number]).columns.tolist()
X_train_encoded = X_train_raw[numeric_cols]
X_test_encoded = X_test_raw[numeric_cols]

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train_encoded)
X_test_scaled = scaler.transform(X_test_encoded)

feature_names = X_train_encoded.columns.tolist()

# ================================
# SECTION 3: Bee Colony Algorithm
# ================================
def run_bee_colony(X_train, X_test, y_train, y_test, feature_names):
    num_features = len(feature_names)
    num_bees = 10
    max_iterations = 10
    limit = 10
    no_improvement_limit = 20

    no_improve_count = 0
    prev_best_accuracy = 0
    best_accuracy = 0
```

```
    best_solution = None

    population = [np.random.choice([0, 1], size=num_features, p=[0.7, 0.3]) for _ in
range(num_bees)]
    fitness = [0] * num_bees
    trial_counter = [0] * num_bees
    convergence = []
    evaluated = {}

    def evaluate(mask):
        key = tuple(mask)
        if key in evaluated:
            return evaluated[key]
        idx = [i for i, bit in enumerate(mask) if bit == 1]
        if not idx:
            evaluated[key] = 0
            return 0
        clf = RandomForestClassifier(n_estimators=30, random_state=np.random.randint(0,
100000))
        acc = cross_val_score(clf, X_train[:, idx], y_train, cv=2, scoring='accuracy', n_jobs=-
1).mean()
        evaluated[key] = acc
        return acc

    # Initial fitness
    for i in range(num_bees):
        fitness[i] = evaluate(population[i])
        if fitness[i] > best_accuracy:
            best_accuracy = fitness[i]
```

```
        best_solution = population[i].copy()

for iteration in range(max_iterations):
    # Phase 1: Employed Bees
    for i in range(num_bees):
        neighbor = population[i].copy()
        flip_idx = np.random.randint(0, num_features)
        neighbor[flip_idx] ^= 1
        acc_new = evaluate(neighbor)
        if acc_new > fitness[i]:
            population[i] = neighbor
            fitness[i] = acc_new
            trial_counter[i] = 0
        else:
            trial_counter[i] += 1
        if fitness[i] > best_accuracy:
            best_accuracy = fitness[i]
            best_solution = population[i].copy()

    # Phase 2: Onlooker Bees
    probs = [f / sum(fitness) if sum(fitness) > 0 else 0 for f in fitness]
    for _ in range(num_bees):
        i = np.random.choice(range(num_bees), p=probs)
        neighbor = population[i].copy()
        flip_idx = np.random.randint(0, num_features)
        neighbor[flip_idx] ^= 1
        acc_new = evaluate(neighbor)
        if acc_new > fitness[i]:
```

```
            population[i] = neighbor
            fitness[i] = acc_new
            trial_counter[i] = 0
        else:
            trial_counter[i] += 1
        if fitness[i] > best_accuracy:
            best_accuracy = fitness[i]
            best_solution = population[i].copy()

    # Phase 3: Scout Bees
    for i in range(num_bees):
        if trial_counter[i] >= limit:
            population[i] = np.random.choice([0, 1], size=num_features, p=[0.7, 0.3])
            fitness[i] = evaluate(population[i])
            trial_counter[i] = 0
            if fitness[i] > best_accuracy:
                best_accuracy = fitness[i]
                best_solution = population[i].copy()

    convergence.append(round(best_accuracy, 4))

    # Early Stopping
    if best_accuracy > prev_best_accuracy:
        prev_best_accuracy = best_accuracy
        no_improve_count = 0
    else:
        no_improve_count += 1
    if no_improve_count >= no_improvement_limit:
```

```
            print(f"Bee Colony stopped at iteration {iteration + 1}")
            break

    idx = [i for i, bit in enumerate(best_solution) if bit == 1]
    selected_features = [feature_names[i] for i in idx]
    return selected_features, idx, convergence, iteration + 1

# ================================
# SECTION 4: Evaluation Utilities
# ================================
def train_and_evaluate(X_train, y_train, X_test, y_test, idx, algorithm_name):
    clf = RandomForestClassifier(n_estimators=30, random_state=np.random.randint(0, 100000),
verbose=1, n_jobs=-1)
    clf.fit(X_train[:, idx], y_train)
    train_preds = clf.predict(X_train[:, idx])
    print(f"\n 🔹 {algorithm_name} - Training Set Evaluation")
    print("Accuracy:", round(accuracy_score(y_train, train_preds), 2))
    print("F1 Score:", round(f1_score(y_train, train_preds), 2))
    return clf

def final_evaluate(clf, X_test, y_test, idx, algorithm_name):
    test_preds = clf.predict(X_test[:, idx])
    probs = clf.predict_proba(X_test[:, idx])[:, 1]
    acc = round(accuracy_score(y_test, test_preds), 2)
    prec = round(precision_score(y_test, test_preds), 2)
    rec = round(recall_score(y_test, test_preds), 2)
    f1 = round(f1_score(y_test, test_preds), 2)
    auc = round(roc_auc_score(y_test, probs), 2)
```

```
    tn, fp, fn, tp = confusion_matrix(y_test, test_preds).ravel()
    print(f"\n✅ {algorithm_name} - Final Evaluation on Test Set")
    print(f"Accuracy: {acc}, Precision: {prec}, Recall: {rec}, F1: {f1}, AUC: {auc}")
    print(f"TP: {tp}, TN: {tn}, FP: {fp}, FN: {fn}")
    return acc, prec, rec, f1, auc, tp, tn, fp, fn

def plot_convergence(convergence, title):
    plt.plot(convergence, marker='o')
    plt.title(f"{title} - Convergence Plot")
    plt.xlabel("Iteration")
    plt.ylabel("Accuracy")
    plt.grid()
    plt.show()

def plot_conf_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
    plt.title(f"{title} - Confusion Matrix")
    plt.grid(False)
    plt.show()

def plot_roc_curve(y_true, y_scores, title):
    RocCurveDisplay.from_predictions(y_true, y_scores)
    plt.title(f"{title} - ROC Curve")
    plt.grid(True)
    plt.show()
```

```
#================================
# SECTION 5: Run Bee Colony
# ================================
bc_features, bc_idx, bc_convergence, bc_iters = run_bee_colony(
    X_train_scaled, X_test_scaled, y_train, y_test, feature_names
)

clf_bc = train_and_evaluate(
    X_train_scaled, y_train, X_test_scaled, y_test, bc_idx, "Bee Colony"
)

acc, prec, rec, f1, auc, tp, tn, fp, fn = final_evaluate(
    clf_bc, X_test_scaled, y_test, bc_idx, "Bee Colony"
)

plot_conf_matrix(y_test, clf_bc.predict(X_test_scaled[:, bc_idx]), "Bee Colony")
plot_roc_curve(y_test, clf_bc.predict_proba(X_test_scaled[:, bc_idx])[:, 1], "Bee Colony")
plot_convergence(bc_convergence, "Bee Colony")

print("\n 🔸 Features selected by Bee Colony:")
print(bc_features)

# ================================
# SECTION 6: Print Summary
# ================================
summary_df = pd.DataFrame({
    'Algorithm': ['Bee Colony'],
    'Accuracy': [acc],
```

```
    'Precision': [prec],
    'Recall': [rec],
    'F1 Score': [f1],
    'AUC': [auc],
    'Selected Features': [len(bc_features)],
    'Iterations': [bc_iters]
})

print(f"\n📊 Bee Colony Performance Summary:")
print(summary_df)
```

Code for Fish Swarm Algorithm

```
# ================================
# SECTION 1: Setup and Imports
# ================================
from google.colab import drive
drive.mount('/content/drive')

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay

# ================================
# SECTION 2: Load and Preprocess Data (Numeric Only)
# ================================
base_path = '/content/drive/MyDrive/IDS_BioInspired_Colab_Final'
data_path = os.path.join(base_path, 'data')
train_file = os.path.join(data_path, 'UNSW_NB15_training-set.csv')
test_file = os.path.join(data_path, 'UNSW_NB15_testing-set.csv')

train_df = pd.read_csv(train_file)
```

```
test_df = pd.read_csv(test_file)

drop_cols = ['label', 'attack_cat']
X_train_raw = train_df.drop(columns=drop_cols)
X_test_raw = test_df.drop(columns=drop_cols)
y_train = train_df['label']
y_test = test_df['label']

X_train_raw = X_train_raw.select_dtypes(include=[np.number])
X_test_raw = X_test_raw.select_dtypes(include=[np.number])

X_all = pd.concat([X_train_raw, X_test_raw])
scaler = MinMaxScaler()
X_all_scaled = scaler.fit_transform(X_all)

X_train_scaled = X_all_scaled[:len(X_train_raw), :]
X_test_scaled = X_all_scaled[len(X_train_raw):, :]
feature_names = X_train_raw.columns.tolist()

# ================================
# SECTION 3: Fish Swarm Algorithm (AFSA Based + Global Early Stop)
# ================================
def run_fish_swarm(X_train, X_test, y_train, y_test, feature_names):
    num_features = len(feature_names)
    num_fish = 10
    max_iterations = 20
    no_improvement_limit = 20
    visual_distance = 5
```

```
    step_size = 1
    crowd_factor = 0.6

    population = [np.random.choice([0, 1], size=num_features, p=[0.7, 0.3]) for _ in
range(num_fish)]
    best_solution = None
    best_accuracy = 0
    prev_best_accuracy = 0
    no_improve_count = 0
    convergence = []
    evaluated = {}

    def evaluate(mask):
        key = tuple(mask)
        if key in evaluated:
            return evaluated[key]
        idx = [i for i, bit in enumerate(mask) if bit == 1]
        if not idx:
            evaluated[key] = 0
            return 0
        clf = RandomForestClassifier(n_estimators=30, random_state=np.random.randint(0,
100000))
        acc = cross_val_score(clf, X_train[:, idx], y_train, cv=2, scoring='accuracy', n_jobs=-
1).mean()
        evaluated[key] = acc
        return acc

    def get_neighbors(index):
        indices = [i for i in range(num_fish) if i != index]
```

```
    np.random.shuffle(indices)
    return [population[i] for i in indices[:visual_distance]]

def move_towards(current, target):
    new = current.copy()
    diff_indices = [i for i in range(num_features) if current[i] != target[i]]
    np.random.shuffle(diff_indices)
    for i in diff_indices[:step_size]:
        new[i] = target[i]
    return new

for iteration in range(max_iterations):
    for i in range(num_fish):
        current = population[i]
        neighbors = get_neighbors(i)
        neighbor_scores = [(evaluate(neighbor), neighbor) for neighbor in neighbors]
        neighbor_scores.sort(reverse=True, key=lambda x: x[0])

        if neighbor_scores:
            best_neighbor_score, best_neighbor = neighbor_scores[0]
            acc_current = evaluate(current)
            if best_neighbor_score > acc_current:
                crowd = sum(np.array_equal(best_neighbor, p) for p in population) / num_fish
                if crowd < crowd_factor:
                    new_position = move_towards(current, best_neighbor)
                else:
                    new_position = np.random.choice([0, 1], size=num_features, p=[0.7, 0.3])
            else:
```

```
                new_position = np.random.choice([0, 1], size=num_features, p=[0.7, 0.3])
            else:
                new_position = np.random.choice([0, 1], size=num_features, p=[0.7, 0.3])

            acc_new = evaluate(new_position)
            acc_old = evaluate(current)
            if acc_new > acc_old:
                population[i] = new_position
                if acc_new > best_accuracy:
                    best_accuracy = acc_new
                    best_solution = new_position.copy()

        convergence.append(round(best_accuracy, 4))

        if best_accuracy > prev_best_accuracy:
            prev_best_accuracy = best_accuracy
            no_improve_count = 0
        else:
            no_improve_count += 1

        if no_improve_count >= no_improvement_limit:
            print(f"Fish Swarm stopped early at iteration {iteration + 1}")
            break

    idx = [i for i, bit in enumerate(best_solution) if bit == 1]
    selected_features = [feature_names[i] for i in idx]
    return selected_features, idx, convergence, iteration + 1
```

```
# ================================
# SECTION 4: Evaluation Utilities
# ================================
def train_and_evaluate(X_train, y_train, X_test, y_test, idx, algorithm_name):
    clf = RandomForestClassifier(n_estimators=30, random_state=np.random.randint(0, 100000), verbose=1, n_jobs=-1)
    clf.fit(X_train[:, idx], y_train)
    train_preds = clf.predict(X_train[:, idx])
    print(f"\n 🔹 {algorithm_name} - Training Set Evaluation")
    print("Accuracy:", round(accuracy_score(y_train, train_preds), 2))
    print("F1 Score:", round(f1_score(y_train, train_preds), 2))
    return clf

def final_evaluate(clf, X_test, y_test, idx, algorithm_name):
    test_preds = clf.predict(X_test[:, idx])
    probs = clf.predict_proba(X_test[:, idx])[:, 1]

    acc = round(accuracy_score(y_test, test_preds), 2)
    prec = round(precision_score(y_test, test_preds), 2)
    rec = round(recall_score(y_test, test_preds), 2)
    f1 = round(f1_score(y_test, test_preds), 2)
    auc = round(roc_auc_score(y_test, probs), 2)
    tn, fp, fn, tp = confusion_matrix(y_test, test_preds).ravel()

    print(f"\n✅ {algorithm_name} - Final Evaluation on Test Set")
    print(f"Accuracy: {acc}, Precision: {prec}, Recall: {rec}, F1: {f1}, AUC: {auc}")
    print(f"TP: {tp}, TN: {tn}, FP: {fp}, FN: {fn}")
    return acc, prec, rec, f1, auc, tp, tn, fp, fn
```

```
def plot_convergence(convergence, title):
    plt.plot(convergence, marker='o')
    plt.title(f"{title} - Convergence Plot")
    plt.xlabel("Iteration")
    plt.ylabel("Accuracy")
    plt.grid()
    plt.show()

def plot_conf_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
    plt.title(f"{title} - Confusion Matrix")
    plt.grid(False)
    plt.show()

def plot_roc_curve(y_true, y_scores, title):
    RocCurveDisplay.from_predictions(y_true, y_scores)
    plt.title(f"{title} - ROC Curve")
    plt.grid(True)
    plt.show()
```

```
# ================================
# SECTION 5: Run Fish Swarm
# ================================
fs_features, fs_idx, fs_convergence, fs_iters = run_fish_swarm(
    X_train_scaled, X_test_scaled, y_train, y_test, feature_names
)

clf_fs = train_and_evaluate(
    X_train_scaled, y_train, X_test_scaled, y_test, fs_idx, "Fish Swarm"
)

acc2, prec2, rec2, f12, auc2, tp2, tn2, fp2, fn2 = final_evaluate(
    clf_fs, X_test_scaled, y_test, fs_idx, "Fish Swarm"
)

plot_conf_matrix(y_test, clf_fs.predict(X_test_scaled[:, fs_idx]), "Fish Swarm")
plot_roc_curve(y_test, clf_fs.predict_proba(X_test_scaled[:, fs_idx])[:, 1], "Fish Swarm")
plot_convergence(fs_convergence, "Fish Swarm")

# ================================
# SECTION 6: Print Features and Summary
# ================================
algorithm_name = "Fish Swarm"
selected_features = fs_features
acc = acc2
prec = prec2
rec = rec2
f1 = f12
auc = auc2
```

```
iters = fs_iters

print(f"\n 🔸 Features selected by {algorithm_name}:")
print(selected_features)

summary_df = pd.DataFrame({
    'Algorithm': [algorithm_name],
    'Accuracy': [acc],
    'Precision': [prec],
    'Recall': [rec],
    'F1 Score': [f1],
    'AUC': [auc],
    'Selected Features': [len(selected_features)],
    'Iterations': [iters]
})

print(f"\n 📊 {algorithm_name} Performance Summary:")
print(summary_df)
```

CODE BOOKLET

SCHOOL OF COMPUTING

2025

CB_V1_2025