Exercise # 5

Consider the following Knapsack Problem.

With a calorie budget of 750 calories, choose an optimal set of foods from the menu.

Food	wine	beer	pizza	burger	fries	coke	apple	donut
Value	89	90	30	50	90	79	90	10
calories	123	154	258	354	365	150	95	195

1. **Write the program code in python that implements TABU SEARCH ALGORITHM to solve the given Knapsack Problem.**

Below is the provided code implementing the Tabu Search Algorithm in Python.

```
import random
# Objective function to minimize
def objective_function(variables, values):
    """
    Calculate the objective value based on the binary variables and their
associated values.

    Parameters:
    variables (list): A list of binary variables (0 or 1).
    values (list): A list of values corresponding to each variable.

    Returns:
    int: The calculated objective value.
    """
    return sum(x * v for x, v in zip(variables, values))
# Constraint function
def constraint(variables, weights, min_calories):
    """
      Check if the given combination of binary variables satisfies the
calorie constraint.
```

```
    Parameters:
    variables (list): A list of binary variables (0 or 1).
     weights (list): A list of weights (calories) corresponding to each
variable.
    min_calories (int): The minimum calorie requirement.

    Returns:
    bool: True if the constraint is satisfied, False otherwise.
    """
    return sum(x * w for x, w in zip(variables, weights)) >= min_calories
# Function to initialize a random binary solution
def initialize_solution(num_variables):
    """
    Initialize a random binary solution.

    Parameters:
    num_variables (int): The number of variables.

    Returns:
    list: A list of random binary values (0 or 1).
    """
    return [random.randint(0, 1) for _ in range(num_variables)]
# Tabu Search algorithm for binary optimization
def    tabu_search_binary_optimization(values,    weights,    min_calories,
max_iterations, tabu_tenure):
    num_variables = len(values)
    current_solution = initialize_solution(num_variables)
    best_solution = current_solution[:]
    best_value = objective_function(current_solution, values)

    tabu_list = []  # Initialize the tabu list

    for _ in range(max_iterations):
        neighbors = []

        for i in range(num_variables):
            neighbor = current_solution[:]
            neighbor[i] = 1 - neighbor[i]
```

```
                if constraint(neighbor, weights, min_calories) and neighbor
not in tabu_list:
                    neighbors.append(neighbor)

            if not neighbors:
                break

             # Aspiration criteria: Allow moves that improve the current best
value
            improving_neighbors = [neighbor for neighbor in neighbors if
objective_function(neighbor, values) < best_value]

            if improving_neighbors:
                neighbor = random.choice(improving_neighbors)
            else:
                neighbor = random.choice(neighbors)

            neighbor_value = objective_function(neighbor, values)

            if neighbor_value < best_value:
                best_solution = neighbor[:]
                best_value = neighbor_value

            current_solution = neighbor[:]

            # Update the tabu list
            if len(tabu_list) >= tabu_tenure:
                tabu_list.pop(0)
            tabu_list.append(neighbor)

    return best_solution, best_value

if __name__ == "__main__":
    # Food items and their respective values and calories
     foods = ["wine", "beer", "pizza", "burger", "fries", "coke", "apple",
"donut"]
    values = [89, 90, 30, 50, 90, 79, 90, 10]
    weights = [123, 154, 258, 354, 365, 150, 95, 195]
    min_calories = 750
    max_iterations = 1000
```

```
    tabu_tenure = 5  # Tabu tenure (you can adjust this value)

     best_solution, best_value = tabu_search_binary_optimization(values,
weights, min_calories, max_iterations, tabu_tenure)

     selected_foods = [foods[i] for i in range(len(best_solution)) if
best_solution[i] == 1]
    total_calories = sum(weights[i] for i in range(len(best_solution)) if
best_solution[i] == 1)

    print("Optimal solution (binary variables):", best_solution)
    print("Selected foods:", selected_foods)
    print("Total calories of selected foods:", total_calories)
    print("Objective value (total value of selected foods):", best_value)
```

2. **Give your final answer. Indicate the stopping criterion, the definition of neighborhood that you used, and values of your parameters.**

```
Optimal solution (binary variables): [0, 0, 1, 1, 0, 0, 0, 1]
Selected foods: ['pizza', 'burger', 'donut']
Total calories of selected foods: 807
Objective value (total value of selected foods): 90
```

Based on the provided code, the best solution involves selecting the items "pizza," "burger," and "donut," resulting in a total value of 90 and a calorie count of 807. The code utilizes a stopping criterion by limiting the number of iterations to 1000. In the code, the neighborhood is defined as a set of binary solutions that can be obtained by flipping a single binary variable (from 0 to 1 or from 1 to 0) in the current solution. So, the neighborhood consists of all the possible binary solutions that can be obtained by making small changes to the current binary solution, one variable at a time. It's a way to explore nearby solutions in the search space to find an optimal or near-optimal solution.

```
    # Food items and their respective values and calories
    foods = ["wine", "beer", "pizza", "burger", "fries", "coke", "apple", "donut"]
    values = [89, 90, 30, 50, 90, 79, 90, 10]
    weights = [123, 154, 258, 354, 365, 150, 95, 195]
    min_calories = 750
    max_iterations = 1000
    tabu_tenure = 5  # Tabu tenure (you can adjust this value)
```

The parameter values used for this optimization problem are provided above and have been used to determine the best possible or close-to-optimal solution.