Nature Inspired Computation
DSAI 403

Assoc. Prof. **Mohamed Maher Ata**
Zewail city of science, technology, and innovation
momaher@zewailcity.edu.eg
Room: S028- Zone D

Why do we still need many different metaheuristic algorithms

Diversity of Problems and Search Landscapes

- Optimization problems vary in structure and difficulty.
- Each algorithm uses unique strategies, some explore globally, others refine locally.
- No single algorithm fits all problem types or landscapes.

No Free Lunch (NFL) Theorem

- States that no algorithm is best for every problem.
- An algorithm that works well on one class will perform worse on another.
- That’s why we need many Metaheuristics, each effective under different conditions.

Key Idea

- Metaheuristics serve as a flexible toolbox of search strategies.
- Each algorithm follows a different search philosophy (e.g., randomness, memory, cooperation).
- Selection or combination depends on the problem’s nature and landscape.
- This diversity drives innovation, hybrid approaches, and adaptability in AI optimization.

Tabu Search (TS)

(Another Story Of Different Metaheuristic Algorithm)

Tabu Search: Human-Inspired Metaheuristic

- Tabu Search is a metaheuristic inspired by **human problem-solving** behavior, specifically the way humans avoid repeating mistakes or previously visited unsuccessful solutions when trying to solve complex problems.
- Humans often remember recent actions and forbid themselves from immediately repeating unsuccessful choices. Tabu Search formalizes this as a “tabu list”, which keeps track of recent moves or solutions to prevent cycling and encourage exploration of new areas in the solution space.
- As a conclusion, Tabu Search is a **memory-based** metaheuristic modeled after the way humans systematically avoid repeating past mistakes to find better solutions.

What is Tabu Search?

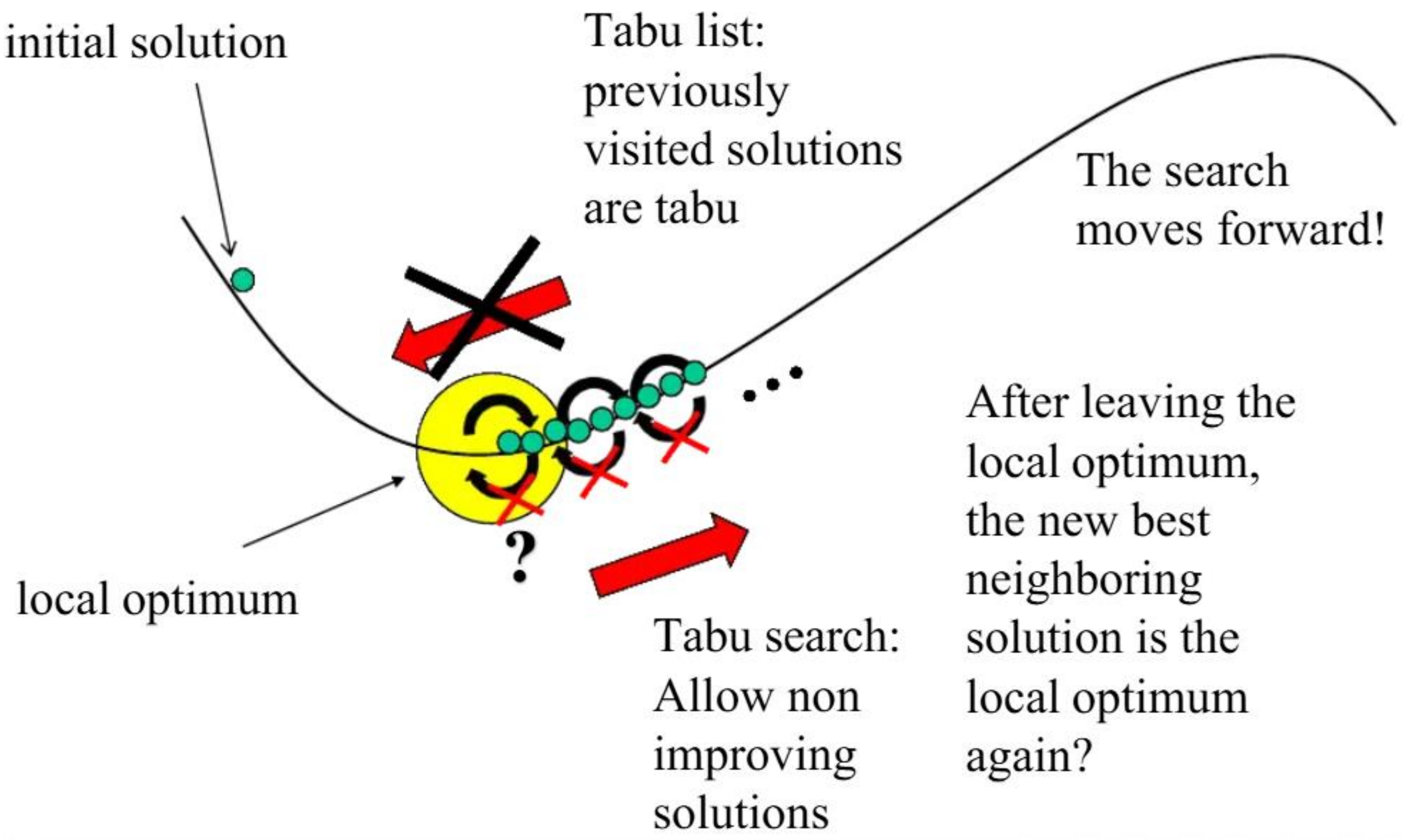
- Tabu Search is an **advanced local search algorithm** that improves upon **Hill Climbing** by **remembering previously visited solutions** (**called *tabu***) to avoid getting stuck in local minima.
- **Main idea:** Don’t just climb up but explore smartly and remember where you have been.
- It Maintains a **Tabu List** of recently visited points to **avoid cycling**. Accordingly, the **Tabu list acts as short-term memory** preventing cycles
- It can **escape local optima** and explore new regions of the search space.
- In Hill Climbing, it climbs up until they can’t go higher. However, in Tabu Search, it remembers previous peaks, sometimes goes down a bit to find a taller one later
- We can say **Hill climbing + short-term memory = Tabu Search**
- It **balances exploration and exploitation**:

  (1) Explores by allowing non-improving moves

  (2) Exploits by always picking the best available neighbor

How Tabu list avoid cycling

- Tabu Search is a **memory-based local search** that guides exploration by forbidding recently visited solutions (tabu) for a few iterations while allowing exceptions (**aspiration**) for globally better moves.



Components of the Tabu Search Algorithm

1	**Solution Representation**	▪ Defines how a solution is encoded for the problem. ▪ Can be a vector of parameters, a set of weights, or an arrangement of items. ▪ Must allow easy modification to generate “neighbors.” ▪ Example: in Deep Learning: a learning rate value η, a dropout rate, or a weight configuration.
2	**Neighborhood Structure**	▪ Defines how new solutions (neighbors) are generated from the current one. ▪ A move changes part of the solution slightly. ▪ Examples: $\eta \pm 0.001$ in learning rate tuning or changing one weight connection in a neural network
3	**Evaluation (Objective Function)**	▪ Measures the quality or cost of each solution. ▪ In optimization, this is what we try to minimize or maximize. ▪ Examples: Deep Learning: validation loss or error rate.
4	**Tabu List (Short-Term Memory)**	▪ Records recent moves or solutions to prevent cycling. ▪ Each stored move has a tabu tenure (number of iterations it remains forbidden). ▪ Updated at every iteration: add new → reduce tenure → remove expired
5	**Tabu Tenure**	▪ Defines how long a move stays in the Tabu List. ▪ Usually between 2 and 7 iterations. ▪ Short tenure: faster revisits, less exploration. ▪ Long tenure: more exploration, slower convergence.
6	**Aspiration Criteria**	▪ Allows a tabu move if it results in a better global solution than any found before. ▪ Prevents missing an excellent solution due to tabu restriction. Break the rule if it improves the best result. ▪ Examples if $\eta = 0.008$ is tabu, but gives loss = $0.22 <$ best(0.23) → accept it and ignore its tabu status

7	**Intensification Strategy (Exploitation)**	▪ Focuses the search near **high-quality solutions** to refine results. ▪ Encourages deeper local exploration around good regions. ▪ **Example:** after finding $\eta = 0.004$ as best, search now nearby $\eta = [0.0035 \text{ and } 0.0045]$
8	**Diversification Strategy (Exploration)**	▪ Forces the search to **move to new regions** when progress stagnates. ▪ Prevents premature convergence to local minima. ▪ **Example:** If search keeps oscillating near x=2 and x=3, jump to x=5 to explore a different region
9	**Stopping Criteria**	▪ Determines **when the algorithm ends.** ▪ **Common options:** (1) Maximum number of iterations reached. (2) No improvement for several iterations. (3) Time limit or desired objective reached. ▪ **In practice:** Tabu Search can stop *naturally* when no admissible (non-tabu) neighbor remain
10	**Memory Structures (For adaptive Tabu only)**	▪ **Short-Term Memory:** tabu list (prevents cycling). ▪ **Intermediate Memory:** stores best solutions (for intensification). ▪ **Long-Term Memory:** tracks visited regions (for diversification).

Core Components

Definition and Background

- Tabu Search (TS) is a metaheuristic optimization method.
- Designed to enhance local search by integrating adaptive memory and strategic decision rules.
- The word "tabu" means forbidden, reflecting how the algorithm prevents cycling back to recent solutions.

Core Concept

- Traditional local search gets stuck in local minima and lacks memory.
- Tabu Search introduces a tabu list (short-term memory) that temporarily forbids recently visited moves.
- This enables the search to accept non-improving moves, avoid repetition, and explore new regions.
- Aspiration criteria allow overriding tabu rules if a move yields a better global solution.

Main Characteristics

- **Adaptive Memory:** Records recent moves to avoid repetition.
- **Aspiration Criteria:** Allows overriding tabu status if a move improves the best-known solution.
- **Flexible Strategy:** Can handle both discrete and continuous problems.
- **Balance of Search:**

  (1) **Intensification***:* deeper search around good solutions.

  (2) **Diversification***:* exploration of new areas when stuck

Example:

We want to minimize: $f(x) = x^2 - 4x + 5$ for $x \in \{0, 1, 2, 3, 4, 5\}$

Assumptions:

- Neighborhood: $N(x) = \{x - 1, x + 1\}$
- Tabu Tenure: 2 iterations
- Initial solution: $x_0 = 4$

Solution:

1. **Solution representation:** $At\ x = 4, f(4) = 5$
2. **Neighborhood structure**: $N(4) = \{3, 5\}$
3. **Evaluation function:** $f(3) = 2, f(5) = 10$, accordingly, best neighbor = 3

Iteration	Current x	f(x)	Tabu List (before)	Neighbors (x→f)	Best admissible move	Move to x	Tabu List after update	Global best
0 (start)	**4**	**5**	ϕ	**3→2 , 5→10**	**3**	**3**	**{4 : 2}**	**(x=3, f=2)**
1	**3**	**2**	**{4 : 2}**	**2→1 , 4→5 (4 tabu)**	**2**	**2**	**{4 : 1, 3 : 2}**	**(x=2, f=1)**
2	**2**	**1**	**{4 : 1, 3 : 2}**	**1→2 , 3→2 (3 tabu)**	**1**	**1**	**{4 : 0 remove, 3 : 1, 2 : 2}**	**(x=2, f=1)**
3	**1**	**2**	**{3 : 1, 2 : 2}**	**0→5 , 2→1 (2 tabu)**	**Stop**	**-**	**{3 : 0 remove, 2 : 1, 1 : 2}**	**(x=2, f=1)**

Note that: **we stop in iteration 3 because:**

- The only neighbor x = 2 is tabu, and the other neighbor x = 0 is worse, giving no improvement.
- Aspiration is not triggered in this example because f(2)=1 equals (not improves) the global best. Hence, no admissible improving (or aspirational) move exists → Stop.

Example:

Maximize f(x) such that:

Assume that:

- initial solution: x_0=0,f(x_0)=3
- Neighborhood: move ±1
- Tabu Tenure = 2 iterations

x	$f(x)$
0	3
1	5
2	4
3	7
4	8

Iteration	Current x	$f(x)$	Tabu List (before)	Neighbors (x→f)	Best admissible move	Move to x	Tabu List after update	Global best
1	0	3	ϕ	1 → 5	1 → 5	1	{0:2}	5
2	1	5	{0:2}	0 → 3, 2 → 4	2 → 4	2	{0:1,1:2}	5
3	2	4	{0:1,1:2}	1 → 5, 3 → 7	3 → 7	3	{0:0 removed, 1:1,2:2}	7
4	3	7	{1:1,2:2}	2 → 4, 4 → 8	4 → 8	4	{1:0 removed, 2:1,3:2}	8
5	4	8	{2:1,3:2}	3 → 7	stop			8

Example:

Suppose we want to tune **one Hyperparameter**; the **learning rate (η)** in order to minimize the loss function of a deep neural network. $L(\eta)=(\eta-0.005)^2+0.002\times\sin(30\eta)$, where, η=[0.001, 0.02]

Assumptions:

- Neighborhood: $\Delta=\pm0.002$
- Tabu Tenure: 2 iterations
- Initial solution: $\eta_0=0.011$

Iteration	Current x	L(η)	Tabu List (before)	Neighbors (η→L)	Best admissible move	Move to η	Tabu List after update	Global best
0	0.011	0.000686	∅	0.009→0.000546	0.009	0.009	{0.011 : 2}	(0.009, 0.000546)
1	0.009	0.000546	{0.011 : 2}	0.007→0.000424 , 0.011→0.000686 (tabu)	0.007	0.007	{0.011 : 1, 0.009 : 2}	(0.007, 0.000424)
2	0.007	0.000424	{0.011 : 1, 0.009 : 2}	0.005→0.00030 , 0.009→0.000546 (tabu)	0.005	0.005	{0.011 : 0 remove, 0.009 : 1, 0.007 : 2}	(0.005, 0.00030)
3	0.005	0.00030	{0.009 : 1, 0.007 : 2}	0.003→0.000184 , 0.007→0.000424 (tabu)	0.003	0.003	{0.009 : 0 remove, 0.007 : 1, 0.005 : 2}	(0.003, 0.000184)
4	0.003	0.000184	{0.007 : 1, 0.005 : 2}	0.001→0.000076 , 0.005→0.00030 (tabu)	0.001	0.001	{0.007 : 0 remove, 0.005 : 1, 0.003 : 2}	(0.001, 0.000076)
5	0.001	0.000076	{0.005 : 1, 0.003 : 2}	0.003→0.000184 (tabu)	Stop (no admissible improving)	-	-	Best η = 0.001 (L=0.000076)

Conclusions:

- Each new step reduced the loss, proving effective exploration.
- Tabu Search stopped intelligently; not by iteration limit, but because no better, non-tabu neighbor exists
- In real time deep learning applications, Even though Tabu Search is designed to stop when no further improvement is possible, we still define a maximum number of iterations as a safety limit. To avoid infinite loops or over-exploration, we add a maximum iteration cap (e.g., 6 or 10 iterations). It guarantees the search terminates in finite time, even if the “no improvement” rule doesn’t trigger cleanly.

Step 1: Load MNIST data and train the model

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
x_train, y_train = x_train[:10000], y_train[:10000]
x_val, y_val = x_test[:2000], y_test[:2000]

def evaluate_model(lr):
    model = keras.Sequential([
        layers.Conv2D(8, 3, activation='relu', input_shape=(28,28,1)),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(32, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    opt = keras.optimizers.Adam(learning_rate=lr)
    model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(x_train, y_train, epochs=1, batch_size=128, verbose=0,
                        validation_data=(x_val, y_val))
    val_acc = history.history['val_accuracy'][-1]
    return val_acc
```

Step 2: Tabu Search setup

```python
tabu_tenure = 2
eta_step = 0.002
eta_min, eta_max = 0.001, 0.020

current_eta = 0.012
tabu_list = {}
history = []

global_best_acc = 0
global_best_eta = current_eta

print("Iter | Current_eta | Val_Acc(%) | Best_Neighbor | Neighbor_Acc(%) | Tabu_List")
print("-----+-------------+------------+----------------+-----------------+-----------------------------")
```

Step 3: Tabu Search Loop

```python
for it in range(1, 10):
    # evaluate current
    current_acc = evaluate_model(current_eta)
    if current_acc > global_best_acc:
        global_best_acc = current_acc
        global_best_eta = current_eta
    # generate neighbors
    neighbors = []
    left = round(current_eta - eta_step, 6)
    right = round(current_eta + eta_step, 6)
    if left >= eta_min:
        neighbors.append(left)
    if right <= eta_max:
        neighbors.append(right)
    # evaluate neighbors with aspiration
    neighbor_scores = []
    for n in neighbors:
        val_acc = evaluate_model(n)
        if n in tabu_list:
            # Aspiration: take tabu neighbor if improves global best
            if val_acc > global_best_acc:
                neighbor_scores.append((n, val_acc))
        else:
            neighbor_scores.append((n, val_acc))
    if not neighbor_scores:
        print(f"{it:4d} | {current_eta:11.6f} | {current_acc*100:10.2f} | (none) | - | {tabu_list}")
        print("\nStopping: No admissible neighbors left.")
        break
```

Step 4: choose best neighbor, update tabu list, and move to next iteration

```
    # choose best neighbor
    best_eta, best_acc = max(neighbor_scores, key=lambda x: x[1])

    # update tabu list: decrement tenures and remove expired
    expired = [k for k, v in tabu_list.items() if v <= 1]
    for k in expired:
        del tabu_list[k]
    for k in tabu_list:
        tabu_list[k] -= 1

    # add current move to tabu list
    tabu_list[round(current_eta, 6)] = tabu_tenure

    # log
    print(f"{it:4d} | {current_eta:11.6f} | {current_acc*100:10.2f} | {best_eta:14.6f} | {best_acc*100:15.2f} | {tabu_list}")

    history.append((it, current_eta, current_acc, best_eta, best_acc, tabu_list.copy()))

    # move to next
    current_eta = best_eta

print("\nFinal chosen learning rate:", current_eta)
print("Global best accuracy:", global_best_acc)
```

Step 5: Results

```
Iter | Current_eta | Val_Acc(%) | Best_Neighbor | Neighbor_Acc(%) | Tabu_List
-----+-------------+------------+---------------+-----------------+-------------------------
   1 |    0.012000 |      92.30 |      0.014000 |           92.20 | {0.012: 2}
   2 |    0.014000 |      93.55 |      0.016000 |           93.05 | {0.012: 1, 0.014: 2}
   3 |    0.016000 |      92.45 |      0.018000 |           92.25 | {0.014: 1, 0.016: 2}
   4 |    0.018000 |      93.65 |      0.020000 |           92.70 | {0.016: 1, 0.018: 2}
   5 |    0.020000 |      92.90 | (none) | - | {0.016: 1, 0.018: 2}

Stopping: No admissible neighbors left.
```

Important note: “Tabu Search stopped at η = 0.020 because no further admissible moves existed, but the best validation accuracy was achieved earlier at η = 0.018 (93.65%).”

Key Differences between Tabu Search and Simulated Annealing, despite both being local search techniques.

Feature	Tabu Search	Simulated Annealing
Memory	Uses a Tabu list to store recently visited solutions or moves.	No memory structure; decisions are based on the current solution and temperature.
Escaping Local Minima	Avoids revisiting recent solutions using Tabu list; diversification is achieved via the memory.	Escapes local minima probabilistically by accepting worse solutions early on (high temperature).
Decision Making	Uses a Tabu list to enforce moves that avoid cycles, making it exploratory.	Accepts worse solutions probabilistically based on temperature, focusing on exploration early on and exploitation later.
Temperature/ Cooling	No temperature concept, uses memory and aspiration criteria to control exploration.	Temperature decreases over time, controlling the exploration-exploitation balance.
Flexibility	More flexible with control over the Tabu list and aspiration criteria.	Less flexible but simpler to implement and understand.
Exploration vs Exploitation	Balances exploration and exploitation with the help of Tabu memory.	Exploration early (high temperature), exploitation later (low temperature).

Types Of Tabu Search (advanced research areas)

Type of Tabu Search	Key Feature	Use Case	Type of Tabu Search
Basic Tabu Search	Uses a Tabu list to avoid revisiting recently explored solutions.	General-purpose optimization	Basic Tabu Search
Adaptive Memory Tabu Search	Dynamically adapts the Tabu list based on search history.	Problems requiring more dynamic memory	Adaptive Memory Tabu Search
Reactive Tabu Search	Tabu tenure changes dynamically based on search progress.	Problems requiring dynamic adjustment	Reactive Tabu Search
Multi-Objective Tabu Search	Handles multiple conflicting objectives with a focus on maintaining Pareto-optimal solutions.	Multi-objective optimization problems	Multi-Objective Tabu Search
Tabu Search with VNS	Variable Neighborhood Search combined with Tabu Search for more diverse exploration.	Problems requiring diversification	Tabu Search with VNS
Tabu Search with Path Relinking	Combines two or more solutions and attempts to find an improved path between them.	Global optimization problems with multiple good solutions	Tabu Search with Path Relinking

Limitations of Tabu Search

Parameter Sensitivity	▪ Performance depends on tabu tenure, neighborhood size, and aspiration criteria. ▪ Poor choice can lead to slow convergence or getting stuck
Memory Management	Maintaining tabu lists and long-term memory structures can become complex and memory-intensive.
No Guaranteed Global Optimum	As a metaheuristic, Tabu Search guides the search intelligently, but it cannot guarantee finding the global optimum
Problem-Specific Tuning	Requires adaptation for each problem type; parameters are not universal.
Computational Cost	Large neighborhoods require evaluating many candidate solutions per iteration.