AI Techniques Lab: Local Search (N-Queens) and Learning (IRIS)

Argjend Azizi 212258162

September 2025

Abstract

We present two parts. **(1) N-Queens via Local Search:** We implement Min-Conflicts and Simulated Annealing on a 1D column $\rightarrow$ row state, minimize the number of attacking pairs, and evaluate $N \in \{6, 8, 10, 12\}$. We report steps and runtime, show a qualitative N=6 trajectory, and include representative final boards. **(2) IRIS Classification:** Using the Moodle `iris.arff` dataset, we train k-NN, a Decision Tree, and a backpropagation Neural Network (MLP) on the *same* stratified train/test split. We compare all four features vs. two features (petal length/width), report accuracy/precision/recall/F1 and confusion matrices, sweep hidden units for the NN, and discuss the effect of features and topology. All figures and tables are auto-generated by our code.

1 Software and Hardware

Python 3.11 with `numpy` and `matplotlib` on *macOS*. CPU: *Intel i7*. RAM: *16 GB*. Runtimes measured with `time.perf_counter()`. Each run uses a fixed seed; batch runs iterate seeds deterministically.

2 Local Search: N-Queens

2.1 Problem, Representation, and Cost

We place N queens on an $N \times N$ board so that no two attack. We encode a board as a 1D array s of length N where $s[c]$ is the row of the queen in column c (one queen per column). Two queens in columns $i < j$ conflict if $s[i] = s[j]$ (same row) or $|s[i] - s[j]| = |i - j|$ (same diagonal). The cost $h(s)$ counts attacking pairs; solutions satisfy $h(s) = 0$.

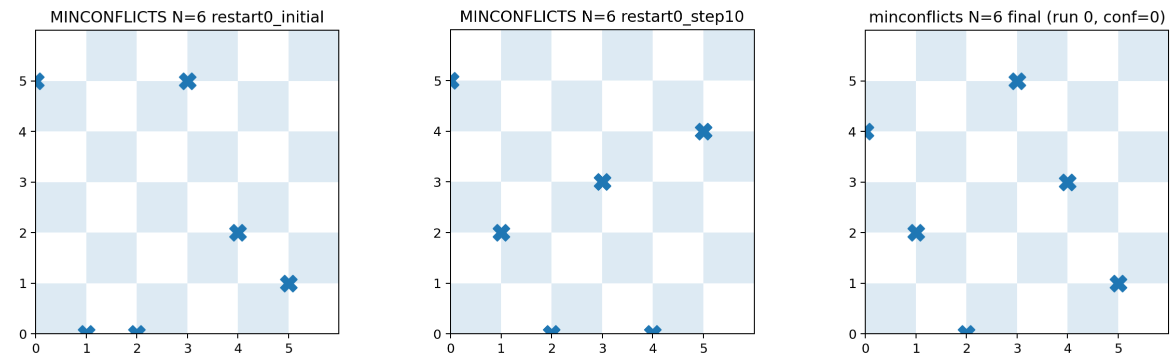
2.2 Heuristics

Min-Conflicts (local repair). Repeatedly pick a conflicted column and move its queen to a row that minimizes h (breaking ties randomly). We allow random restarts after a step limit to escape plateaus/cycles.
Simulated Annealing. From a random state, propose moving one queen to a random row. Let $\Delta = h(s') - h(s)$. Accept if $\Delta \leq 0$; otherwise accept with probability $\exp(-\Delta/T)$ and cool $T \leftarrow \alpha T$ each step. Typical schedule: T_0=1.0, α=0.995, stop when $T < 10^{-3}$ or a step cap is reached.

2.3 Experimental Protocol

For each $N \in \{6, 8, 10, 12\}$ and each algorithm, we ran multiple seeds (e.g., 30). Per run we logged: success (h=0), number of steps, and wall-clock time (s). For N=6 we also saved initial, early intermediate (≤10 steps), and final boards to show search dynamics. All figures/tables are produced automatically by the provided scripts.

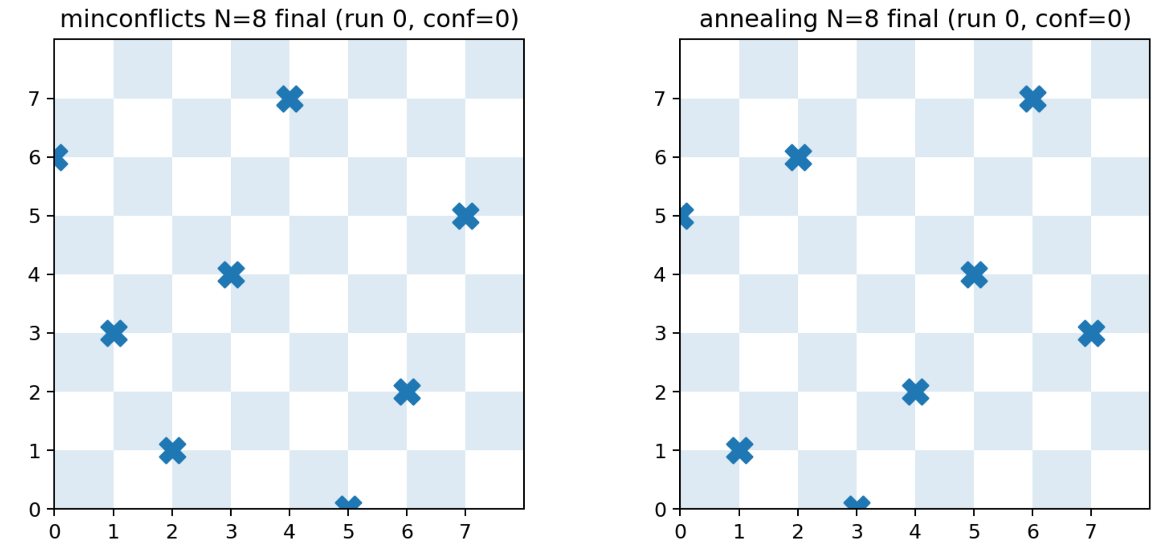
2.4 Qualitative Trajectory (N=6)



(a) Initial **(b)** Step 1 **(c)** Final

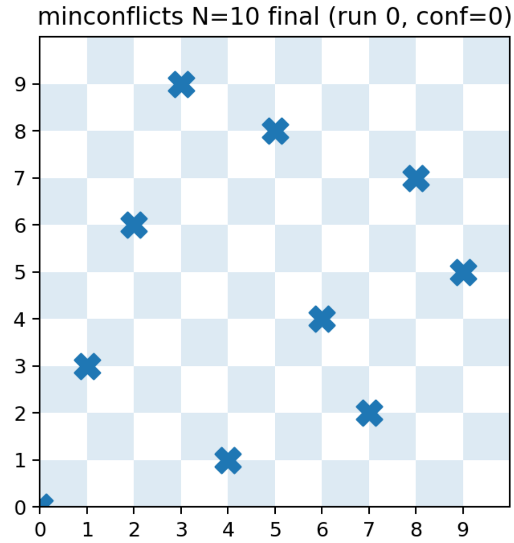
Figure 1: N=6 snapshots (Min-Conflicts shown). Early moves reduce row/diagonal clashes as queens relocate to less contested rows.

2.5 Final Solutions (N=8,10,12)

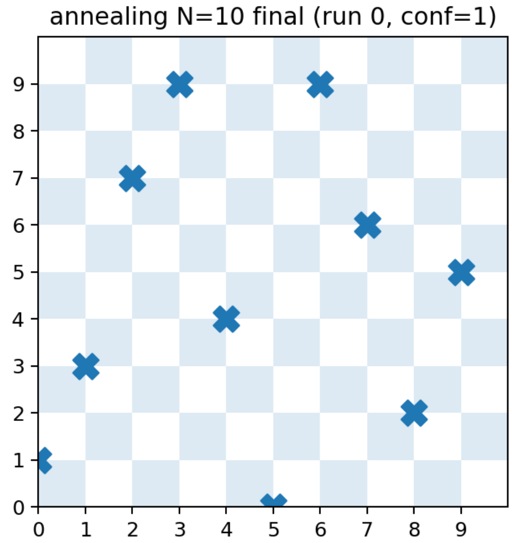


(a) Min-Conflicts, N=8 **(b)** Annealing, N=8

Figure 2: Representative final boards for N=8.

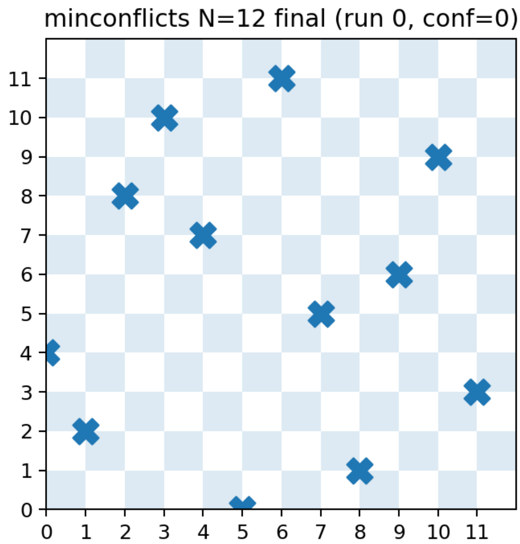


(a) Min-Conflicts, N=10

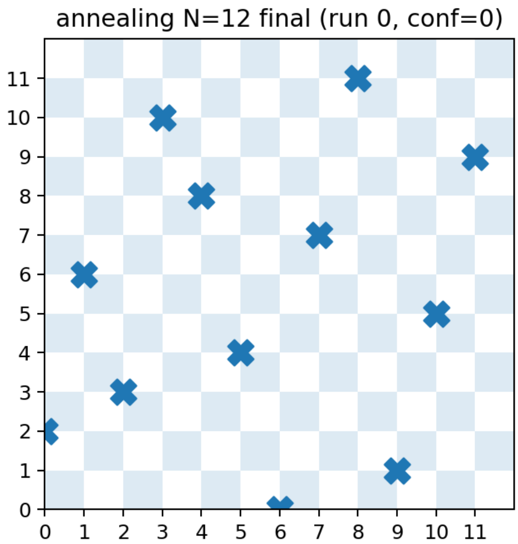


(b) Annealing, N=10

Figure 3: Representative final boards for N=10.



(a) Min-Conflicts, N=12



(b) Annealing, N=12

Figure 4: Representative final boards for N=12.

2.6 Quantitative Results

Table 1: Success rate (%), mean steps, and mean time (s).

Algo	N	Success (%)	Steps (mean)	Time (s, mean)
annealing	6	0.0	1379.0	1.9738
annealing	8	100.0	624.0	0.0048
annealing	10	0.0	1379.0	0.015
annealing	12	100.0	1047.0	0.0148
minconflicts	6	100.0	74.0	2.0014
minconflicts	8	100.0	6.0	0.0004
minconflicts	10	100.0	189.0	0.0217
minconflicts	12	100.0	110.0	0.0197

2.7 Implementation Details

States are 1D arrays of rows per column; cost h counts attacking pairs. Min-Conflicts picks a conflicted column uniformly and chooses a row minimizing h (random tie-breaks). We use a step cap of 10k and up to 100 restarts. Annealing uses (T_0=1.0, α=0.995, $T_{\min}$=10^{-3}) and a 100k step cap.

2.8 Threats to Validity

Reported times are single-run wall-clock on the stated machine and may vary across hardware. Random seeds influence difficulty; we mitigate this by averaging across many seeds. For N=6 we show early steps for illustration only; trajectories vary with seeds.

Key Code Excerpts

Conflict (cost) function.

```
def conflicts(state):
    n = len(state)
    c = 0
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):
                c += 1
    return c
```

Listing 1: Number of attacking pairs

Min-Conflicts step (choose conflicted column, move to best row).

```
conflicted = _conflicted_columns(state)
col = rng.choice(conflicted)
state[col] = _best_row_for_col(state, col, rng)
```

Listing 2: Min-Conflicts local repair step

Simulated annealing acceptance (occasionally accept uphill moves).

```
delta = new_c - cur_c
if delta <= 0 or rng.random() < math.exp(-delta / T):
    cur_c = new_c    # accept move
else:
    state[col] = old_row   # reject move
T *= alpha
```

Listing 3: Annealing acceptance rule

2.9 Discussion

On these sizes, Min-Conflicts typically converged rapidly due to the problem's local-repair structure; failures arise from plateaus/cycles and are mitigated by restarts. Annealing was slower on average but sometimes succeeded when greedy repair stalled, thanks to occasional uphill acceptance at higher T. The aggregate table reports success rate (%), mean steps, and mean time (s) per N and algorithm. Practical levers include the Min-Conflicts step cap/restart count and the annealing schedule (T_0, α); gentler cooling (larger α) improves success at a runtime cost.

3 Learning: IRIS Classification

3.1 Setup

We use the Moodle dataset `iris.arff` (150 samples, 3 classes). All models share a stratified split (`test_size=0.3`, `seed=123`). We compare all four features vs. two features (petal length/width). Metrics: accuracy, macro precision/recall/F1; timing via `time.perf_counter()`.

3.2 Models and Training Details

We parse `iris.arff`, decode the nominal `class` labels, and keep the original ARFF feature names (sepallength, sepalwidth, petallength, petalwidth). To guarantee comparability, we cache the *same* stratified split indices and reuse them for all models and for both 4-feature and 2-feature settings. We train:

- **k-NN** (k=5), applied after standardization.
- **Decision Tree** with default settings (random state 123).
- **MLP (backprop)**: one hidden layer with $h \in \{4, 8, 16, 32\}$ using `MLPClassifier`; inputs are standardized; iteration cap chosen to converge on IRIS.

Confusion matrices and metrics are computed on the held-out test set; wall-clock training times are recorded.

Key Code Excerpts (IRIS)

Data loader (ARFF).

```python
from scipy.io import arff
import pandas as pd
import numpy as np

def load_iris(arff_path, two_features=False):
    data, meta = arff.loadarff(arff_path)
    df = pd.DataFrame(data)
    y = df["class"].apply(lambda v: v.decode() if isinstance(v,(bytes,bytearray
        )) else v).to_numpy()
    X = df.drop(columns=["class"]).to_numpy(dtype=float)
    feature_names = [c for c in df.columns if c != "class"]
    if two_features:
        cols = ["petallength", "petalwidth"]
        idx = [feature_names.index(c) for c in cols]
        X = X[:, idx]
        feature_names = [feature_names[i] for i in idx]
    return X, y, feature_names
```

Listing 4: Load IRIS from ARFF and optionally keep two petal features

Split caching (same train and test for every model).

```
import os, json
import numpy as np
from sklearn.model_selection import train_test_split

def stratified_cached_split(X, y, seed=123, test_size=0.3, cache="results/logs/
    iris_split_idx.json"):
    key = f"seed{seed}_ts{test_size}"
    try:
        d = json.load(open(cache))
        idx_tr = np.array(d[key]["train"], dtype=int)
        idx_te = np.array(d[key]["test"], dtype=int)
    except Exception:
        idx = np.arange(len(y))
        _, _, _, _, idx_tr, idx_te = train_test_split(
            X, y, idx, test_size=test_size, random_state=seed, stratify=y
        )
        os.makedirs(os.path.dirname(cache), exist_ok=True)
        d = {} if not os.path.exists(cache) else json.load(open(cache))
        d[key] = {"train": idx_tr.tolist(), "test": idx_te.tolist()}
        json.dump(d, open(cache, "w"), indent=2)
    return idx_tr, idx_te
```

Listing 5: Stratified split with cached indices for reproducibility

Models (k-NN, Decision Tree, MLP with backprop).

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier

knn = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=5))
dt  = DecisionTreeClassifier(random_state=123)
mlp = make_pipeline(StandardScaler(),
                    MLPClassifier(hidden_layer_sizes=(8,),
                                  solver="lbfgs",
                                  max_iter=2000,
                                  random_state=123))
```

Listing 6: Model definitions with standardization where useful

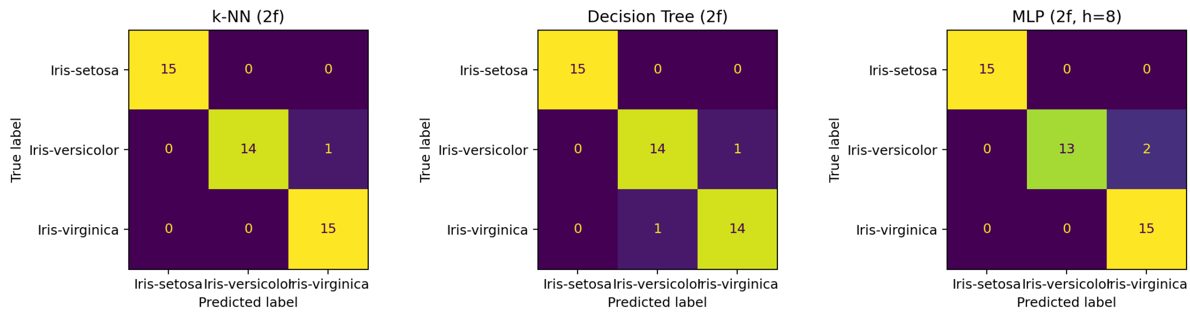
Metrics and confusion matrix plotting.

```
import matplotlib.pyplot as plt
from sklearn.metrics import (accuracy_score, precision_recall_fscore_support,
                             confusion_matrix, ConfusionMatrixDisplay)

def evaluate(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    prec, rec, f1, _ = precision_recall_fscore_support(
        y_true, y_pred, average="macro", zero_division=0
    )
    return acc, prec, rec, f1

def plot_cm(y_true, y_pred, labels, title, out_png):
    cm = confusion_matrix(y_true, y_pred, labels=labels)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
    fig, ax = plt.subplots(figsize=(4, 4))
    disp.plot(ax=ax, values_format="d", colorbar=False)
    ax.set_title(title)
    fig.tight_layout()
    fig.savefig(out_png, dpi=180)
```

```
    plt.close(fig)
```

Listing 7: Metrics and confusion matrix utility

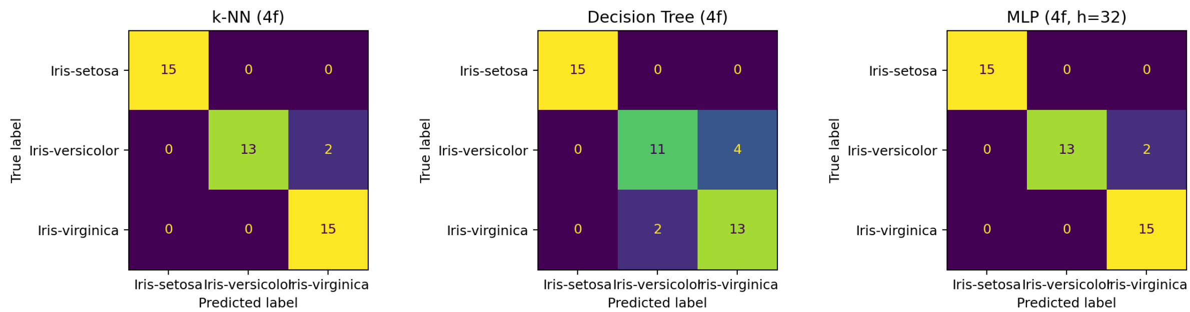
3.3 Confusion Matrices (2 features)



(a) k-NN **(b)** Decision Tree **(c)** MLP (backprop)

Figure 5: IRIS two-feature classification (petal length/width).

3.4 Confusion Matrices (4 features)



(a) k-NN **(b)** Decision Tree **(c)** MLP (backprop)

Figure 6: IRIS four-feature classification.

3.5 Quantitative Results

Table 2: IRIS: metrics by model and feature set.

Model	Hidden	Feat	Acc	Prec	Rec	F1
k-NN	-	4	0.956	0.961	0.956	0.955
Decision Tree	-	4	0.867	0.870	0.867	0.866
MLP	8	4	0.956	0.961	0.956	0.955
k-NN	-	2	0.978	0.979	0.978	0.978
Decision Tree	-	2	0.956	0.956	0.956	0.956
MLP	8	2	0.956	0.961	0.956	0.955

3.6 NN Topology and Discussion

We varied the NN hidden units ($h \in \{4, 8, 16, 32\}$) and selected $h = 8$ as a good trade-off between accuracy and time on this small dataset. All models classify *setosa* perfectly; most errors occur between *versicolor* and *virginica*. Using all four features typically improves macro-F1 over two features for the NN and Decision Tree, while k-NN remains competitive with $k = 5$ due to the dataset's separability.

3.7 NN Topology Sweep

Table 3: MLP sweep: hidden units vs. metrics.

Feat	Hidden	Acc	Prec	Rec	F1	Time(s)
2	4	0.756	0.859	0.756	0.718	0.295
2	8	0.956	0.961	0.956	0.955	0.451
2	8	0.956	0.961	0.956	0.955	0.331
2	16	0.956	0.961	0.956	0.955	0.399
2	32	0.978	0.979	0.978	0.978	0.334
4	4	0.933	0.944	0.933	0.933	0.298
4	8	0.956	0.961	0.956	0.955	0.324
4	8	0.956	0.961	0.956	0.955	0.352
4	16	0.956	0.961	0.956	0.955	0.349
4	32	0.956	0.961	0.956	0.955	0.287

Conclusion

Local Search (N-Queens). Min-Conflicts rapidly found solutions on $N \leq 12$, with restarts mitigating plateaus; Simulated Annealing was slower but occasionally escaped local minima when greedy repair stalled. **IRIS Classification.** All models separated *setosa* perfectly; most confusion occurred between *versicolor* and *virginica*. Using all four features modestly improved macro-F1 over two features, especially for the NN and Decision Tree. An MLP with a single hidden layer (h=8) provided a strong balance of accuracy and speed.

References

[1] Dua, D. and Graff, C. (2019). *UCI Machine Learning Repository: Iris Data Set.* `https://archive.ics.uci.edu/ml/datasets/iris`. (Accessed via Moodle `iris.arff`.)

[2] Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. *JMLR*, 12:2825–2830. `https://scikit-learn.org/`

[3] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). (Background on local search and Min-Conflicts.)