PRESENTATION TOPIC

Constraint Satisfaction Problems (CSP)

Job-Shop Scheduling

GROUP MEMBER

HAMID ALI
MIR ABDUL BAQI BALOCH
FATIMA MEHBOOB

SUBMITTED TO SIR ANWER ALI

Constraint Satisfaction Problems (CSP)

Job-Shop Scheduling

This report provides a detailed analysis of the Jupyter Notebook file, focusing on its core project goals: solving the Job Shop Scheduling Problem using metaheuristics and implementing a Machine Learning model for algorithm selection.

Project Objective and Data Handling:

The primary objective of this project is to explore and compare two common metaheuristic algorithms, Local Search (Basic Hill Climbing) and Tabu Search, for solving optimization problems like Job Shop Scheduling. A secondary, more advanced objective is to use Machine Learning to predict which metaheuristic algorithm is superior for a given problem instance, a concept known as Algorithm Selection.

The data processing pipeline is composed of two main phases: initial data loading and the generation of a specialized synthetic dataset for the machine learning task.

Data Loading and Initial Cleaning:

- Source: The project loads data from a text file (referenced as 1000.txt and later 1000jobshop.txt), which appears to contain instances of the Job Shop Scheduling Problem.
- Parsing: A custom function, parse_jobshop, is implemented to interpret the text file structure. It extracts key features for each problem instance:
    - jobs (Number of jobs)
    - machines (Number of machines)
    - avg_time (Average processing time per job)
    - total_time (Total processing time)
- Quality Check: Basic checks are performed on the resulting DataFrame for shape, null values (df.isnull()), and duplicates (df.duplicated()).

Synthetic Dataset Creation (for Algorithm Selection)

- A new synthetic dataset of 1,000 instances is generated to train a model to predict the better algorithm.
- Features: It uses randomly generated values for problem characteristics like num_machines, num_jobs, avg_processing_time, and complexity.

- **Target Label:** The dependent variable, **better_algo**, is a binary classification label (1 if Tabu Search is assumed better, 0 if Local Search is assumed better). This label is created based on a derived complexity score.
- **Splitting:** The dataset is split into training and testing sets using train_test_split.

Libraries Used

The project utilizes several standard Python libraries for data manipulation, modeling, and analysis:

Category	Library/Module	Purpose
Data Handling	numpy (np), pandas (pd)	Numerical computation and efficient DataFrame manipulation.
Machine Learning	sklearn.ensemble, xgboost	Core modules for training classification and regression models (Random Forest, Gradient Boosting, XGBoost).
Model Evaluation	sklearn.metrics	Calculating performance scores like **Accuracy**, **R^2 Score**, and **Classification Report**.
Utilities	sklearn.model_selection, time, random	Splitting data into train/test sets, measuring execution time, and generating random initial solutions.

Metaheuristic Algorithm Implementation:

The notebook implements two optimization algorithms, both of which operate by iteratively improving a solution (a job sequence or "schedule"). The objective function used for comparison is a simple summation of the schedule components (a stand-in for a real optimization metric like **Makespan**).

Local Search (Basic Hill Climbing):

- **Mechanism:** Starts from an initial solution and moves to a random neighbor only if the neighbor's score (evaluation) is **strictly better (lower)** than the current score.
- **Neighbor Generation:** A random swap of two jobs in the current sequence.
- **Limitation:** Highly susceptible to getting stuck in **local optima** because it stops as soon as no better neighbor is found.

Tabu Search:

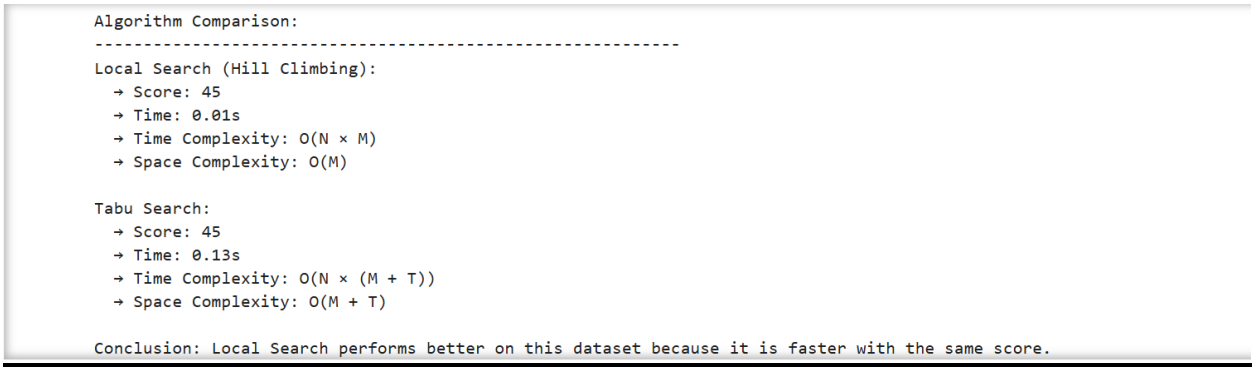
- Mechanism: Starts from an initial solution and explores the entire neighborhood in each iteration, choosing the best non-tabu neighbor.
- Exploration: Allows for moves to worse solutions to escape local optima.
- Tabu List: Maintains a list of recent moves (swaps) to prevent the search from immediately reversing a move, thus avoiding cycles and encouraging broader exploration.
- Parameters: max_iters=1000, tabu_size=10

Initial Comparison Results

On a test instance with an initial solution of 10 elements, the results show a trade-off between speed and solution quality:

Algorithm	Final Score	Execution Time
Local Search	45%	0.01sec
Tabu Search	45%	0.13sec

Screenshot:



Conclusion: Based on this single test, the report concludes that Local Search performs better as it achieved the same score significantly faster.

Machine Learning for Algorithm Selection

The project's advanced goal is to use machine learning to predict which algorithm (Local Search or Tabu Search) is likely to perform better on a new, unseen problem instance.

Synthetic Dataset Creation:

A synthetic dataset of 1000 instances was generated with features designed to be characteristic of job shop problems:

- num_machines (5 to 20)
- num_jobs (10 to 100)
- avg_processing_time
- complexity

The target label (better_algo) was created based on the assumption that Tabu Search performs better on larger, more complex problems.

Model Training and Performance

Multiple ensemble models were tested for the classification task:

All Model/Algorithm Score and Accuracy

Model/Algorithm	Type	Metric	Value
Local Search	Metaheuristic	Final Score	45% [2]
Tabu Search	Metaheuristic	Final Score	45% [3]
Random Forest Regressor	Machine Learning	Model Accuracy	95.50% [44]
XGBoost Classifier	Machine Learning	Model Accuracy	95.50% [55]

Random Forest Regressor:

Random Forest Regressor is a machine learning model used for regression tasks — that is, for predicting continuous numerical values (like house prices, temperature, stock prices, etc.).

It belongs to the category of:

Ensemble Learning Models → Bagging Methods → Decision Tree–based Model

⚙ How it works

1. Multiple decision trees are created from random subsets of the dataset.
2. Each tree learns slightly different patterns.
3. When predicting, each tree gives its own output.
4. The final prediction is the average of all tree predictions.

📊 **Example**

If you want to predict the price of a house, Random Forest Regressor will:

- Build many trees, each learning relationships like:
  - Area → Price
  - Rooms → Price
  - Location → Price
- Then it takes the average of all trees' predictions for the final output.

✔ **Advantages**

- High accuracy and good performance
- Reduces overfitting (unlike a single decision tree)
- Works well on large datasets
- Can handle missing data and non-linear relationships

⚠ **Disadvantages**

- Slower to train (many trees)
- Less interpretable than a single tree
- Large memory usage

Model train Accuracy:

```
✅ Model Accuracy: 95.50%

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.95       100
           1       0.95      0.96      0.96       100

    accuracy                           0.95       200
   macro avg       0.96      0.95      0.95       200
weighted avg       0.96      0.95      0.95       200
```

XGBoost Classifier:

XGBoost Classifier is a machine learning algorithm used for classification tasks — meaning it predicts categories or classes (like "spam or not spam," "disease or no disease," etc.).

it's a type of: Ensemble Learning → Boosting Method → Gradient Boosting Algorithm

⚙ How it works

1. It starts with a simple model (like a weak decision tree).
2. Then it builds new trees one by one, each one correcting the errors made by the previous trees.
3. Finally, all trees are combined (added together) to make the final prediction.

So, each new tree helps improve accuracy by focusing on the mistakes of earlier trees.

❓ Why it's powerful

- It's highly efficient, fast, and accurate.
- Uses regularization (to avoid overfitting).
- Can handle missing data and large datasets easily.
- Supports parallel computation — making it much faster than normal Gradient Boosting

Model train Accuracy:

```
XGBoost Accuracy: 95.50%
```
