Code Implementation Explanation

This document explains the implementation of the graph search task using Python. The solution is split across two main files: `graph_search_task.py` and `helper.py`.

File Structure

- `graph_search_task.py`: Main script that executes the graph search task
- `helper.py`: Contains utility functions and core functionality
- Supporting files:
  - `feature_graph.json`: Pattern graph data
  - `workpiece_graph.json`: Main workpiece graph data
  - Generated visualizations (*.html files)

Helper.py Explanation

Contains core functionality split into four main sections:

1. JSON Loading (`load_json`)

- Handles loading and parsing of JSON files
- Includes error handling for:
  - Missing files
  - Invalid JSON format
- Returns parsed JSON data or None if errors occur

2. Graph Creation (`create_graph`)

- Converts JSON data into NetworkX graph objects
- Handles:
  - Node creation with attributes
  - Edge creation with attributes
- Supports the specific format of the input JSON files

3. Visualization Functions

Two visualization approaches are implemented:

2D Visualization (`visualize_graph`)

- Creates basic interactive graph view
- Uses color coding:
  - Lightblue for cavity nodes
  - Lightgreen for non-cavity nodes
- Shows node types and edge relationships

3D Visualization (`visualize_graph_3d`)

- Creates enhanced 3D interactive view
- Features:
  - Dark mode interface
  - Custom node shapes based on type:
    - Plane: Circle (Green)
    - Cylinder: Box (Orange)
    - Cone: Triangle (Red)
    - Torus: Diamond (Blue)
  - Physics-based layout
  - Enhanced edge visibility

4. Graph Matching Functions

Three matching functions for subgraph isomorphism:

- `strict_node_match`: Matches nodes on type AND cavity status
- `relaxed_node_match`: Matches nodes on type only
- `edge_match`: Matches edges based on angular type

graph_search_task.py Explanation

Main script that implements the search task in four sections:

1. Data Loading

- Loads both feature and workpiece graphs
- Uses helper functions to parse JSON data

2. Graph Creation

- Creates NetworkX graph objects
- Generates both 2D and 3D visualizations

3. Subgraph Matching

Implements two types of matching:

- Strict matching (exact matches)
- Relaxed matching (pattern matches)

4. Results Output

Prints comprehensive results including:

- Match status for both strict and relaxed criteria
- Number of matches found
- Detailed match information

Usage

1. Ensure all dependencies are installed:

```
pip install -r requirements.txt
```

2. Run the script:

```
python graph_search_task.py
```

3. The script will generate 4 HTML files:

- `feature_graph.html`: 2D visualization of the feature graph
- `feature_graph_3d.html`: 3D visualization of the feature graph
- `workpiece_graph.html`: 2D visualization of the workpiece graph
- `workpiece_graph_3d.html`: 3D visualization of the workpiece graph

4. The script will also print the results to the console.

Implementation Notes

Steps Taken To Implement The Solution

1. Forked the repository and cloned it to my local machine.
2. Created a virtual environment and installed the dependencies.
3. Load the JSON data using the `load_json` function.
4. Created the feature and workpiece graphs using the `create_graph` function.
5. Generated the 2D and 3D visualizations using the `visualize_graph` and `visualize_graph_3d` functions.

6. Implemented the strict and relaxed matching functions using the `strict_node_match`, `relaxed_node_match`, and `edge_match` functions.
7. Printed the results to the console.
8. Pushed the changes to the branch.

Graph Matching

The graph matching is implemented using the `nx.algorithms.isomorphism.GraphMatcher` class.

Strict Matching

1. The strict matching is implemented using the `strict_node_match` and `edge_match` functions.
2. We match both the type and the cavity status along with the edge angular type.

Relaxed Matching

1. The relaxed matching is implemented using the `relaxed_node_match` and `edge_match` functions.
2. We match only the type along with the edge angular type.