ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

Examination Schedule Management System for Faculty Members at Tay Do University

Trinh Quang Minh[1]; Ngo Thi Lan[2]

[1]FSB Institute of Management & Technology, FPT University
[2] Faculty of Engineering - Technology, Tay Do University, Can Tho City, Viet Nam.

Publication Date: 2026/01/28

Abstract: This paper presents the design and implementation process of an Exam Invigilation Scheduling Management System for lecturers at Tay Do University. The system automates the assignment of invigilators based on real-world data collected during the period 2021–2025. The core solution of the research is the combination of Graph Coloring algorithms to handle time conflicts and Greedy heuristics algorithms to ensure fairness in workload allocation. Exam sessions are modeled as nodes in the graph, where edges represent scheduling conflicts (matching dates and times). The system has been deployed on the Kaggle platform with interactive dashboards, allowing for transparent data analysis and visualization. Experimental results show that the system is capable of: Automatically detecting and eliminating 100% of scheduling conflicts between lecturers and exam rooms. Balancing workloads across departments helps reduce standard deviation in task allocation. Optimizing human resources through faculty rotation using modulo operations. Providing an intuitive query interface by date, class, or faculty member enhances educational management efficiency. This research contributes a data-driven approach, transitioning from manual management processes to intelligent automation systems, suitable for the practical context of Vietnamese universities.

Keywords: *Greedy Algorithm, Graph Coloring Algorithm, Optimization, Scheduling, Invigilation, Data Visualization*.

How to Cite: Trinh Quang Minh; Ngo Thi Lan (2026) Examination Schedule Management System for Faculty Members at Tay Do University. *International Journal of Innovative Science and Research Technology*, 11(1), 2045-2052. https://doi.org/10.38124/ijisrt/26jan518

I. INTRODUCTION

Organizing and managing exam invigilation schedules is one of the most important and complex administrative tasks at higher education institutions. At Tay Do University, a key training institution in the Mekong Delta region, the volume of exams and the number of invigilators are very large. However, the current assignment of invigilation schedules still relies heavily on manual processes, leading to several significant challenges: Errors and conflicts: Manual scheduling easily leads to scheduling conflicts (a lecturer being assigned to two different exam rooms at the same time). Unbalanced workload: It is difficult to ensure absolute fairness in the number of invigilation shifts among lecturers and departments. Lack of intuitiveness: Administrators have difficulty monitoring the overall situation and quickly retrieving data when unexpected changes occur. This research aims to design and implement a data-driven Exam Invigilation Schedule Management System to thoroughly address these issues. The system integrates modern optimization algorithms such as Graph Coloring and Greedy algorithms to automate the assignment process. The main contributions of this paper include: Building a mathematical model: Transforming the staff assignment problem into a graph coloring problem, where exam sessions are nodes and time conflicts are edges. Resource optimization: Using Modulo operations to rotate faculty, ensuring fair and efficient allocation. Data visualization: Developing interactive dashboards on the Kaggle platform, enabling transparent management and analysis of the actual exam schedule from 2021–2025. The research results not only solve a practical problem at Tay Do University but also provide a reference framework for applying data science to educational management in Vietnam.

II. METHODOLOGY & RELATED WORK

Previous research on exam scheduling has focused on: Timetabling algorithms in universities (graph coloring, constraint satisfaction). Fair workload distribution using greedy heuristics. Visualization dashboards for non-technical users. However, few studies apply these techniques directly to Vietnamese universities with real datasets. This paper contributes by adapting graph theory and greedy algorithms to the context of Tay Do University.

- *Dataset*

Four cleaned Excel datasets (2021–2025) containing exam schedules, invigilator assignments, and departmental summaries.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

- *Key Tables*: Exam Schedule, Exam Invigilators, CBCT Summary.

➢ *System Design*

- *Data preprocessing*: cleaning, normalization, merging sheets.
- *Conflict detection*: checking overlapping exams by date, time, and room.
- *Graph Coloring Algorithm*: each exam slot is a node; edges represent conflicts; colors represent safe invigilator groups.
- *Greedy Algorithm*: ensures fair distribution of invigilators across departments.
- *Dashboard (Gradio/Plotly):* interactive visualization of schedules and assignments.

➢ *Implementation*

- Environment: Kaggle Notebook (Python, Pandas, NetworkX, Plotly).
- Code repositories: Main system: Kaggle Notebook (kaggle.com)
- Graph Coloring Algorithm: Kaggle Notebook (kaggle.com)

➢ *Some Notable Studies:*

- AI-Based Intelligent System for Personalized Examination Scheduling. Authors: Marco Barone, Muddasar Naeem, Matteo Ciaschi, Giancarlo Tretola, Antonio Coronato. Content: Applying artificial intelligence to build a personalized exam scheduling system for students. Source: MDPI Technologies, 2025
- Cornell University Uses Integer Programming to Optimize Final Exam Scheduling. Authors: Tinghan Ye, Adam S. Jovine, Willem van Osselaer, Qihan Zhu, David B. Shmoys. Content: Using integer programming to optimize final exam scheduling at Cornell University. Source: arXiv, 2024
- University Exam Scheduling System Using Graph Coloring Algorithm and RFID Technology. Authors: Akhan Akbulut, Güray Yılmaz. Content: Applying a graph coloring algorithm combined with RFID to solve the exam scheduling problem. Source: IJIMT
- Exam Timetabling Using Graph Coloring Approach. Authors: Burairah Hussin, Abd Samad Hasan Basari, Abdul Samad Shibghatullah, Siti Azirah Asmai. Content: Research on applying a graph coloring algorithm to the exam scheduling problem at universities. Source: CORE Repository, UTeM Malaysia
- Modeling and Optimization of the Exam Invigilator Assignment Problem Based on Preferences. Content: Modeling and optimizing the assignment of invigilators based on preferences and constraints. Source: Academia.edu

III. RESULTS AND DISCUSSION

The implementation of the proposed system yielded significant results across different academic terms. During the HK2 2023–2024 period, the system successfully processed 635 exam slots over 16 days, while for HK3 2024–2025, it managed 72 slots across 4 days. The system's conflict detection module effectively identified overlaps in both invigilator assignments and room utilization. By applying a Graph Coloring algorithm, the model generated non-conflicting proctor groups, while the Greedy Algorithm ensured a balanced workload distribution, significantly reducing the standard deviation of assignments. Furthermore, integrated visualization dashboards allowed faculty members to efficiently query schedules by date, class, or lecturer.

➢ *Statistics:*

- HK2 2023–2024: 635 exam slots across 16 days.
- HK3 2024–2025: 72 exam slots across 4 days.

- Conflict Detection: identified overlapping invigilator assignments and room usage.
- Graph Coloring Output: produced non-conflicting invigilator groups.
- Greedy Algorithm Output: balanced workload distribution, reducing standard deviation of assignments.
- Visualization: dashboards enabled faculty to query schedules by date, class, or lecturer.

➢ *Introduction to the Graph Coloring Algorithm*

In this study, the problem of assigning invigilators is modeled as a graph coloring problem. The goal is to optimize personnel allocation to avoid time conflicts and ensure fair distribution. This flowchart helps you clearly visualize how the system operates from data input → algorithm processing → task assignment → verification → result output.

- *General Process Flowchart (System Workflow)*

✓ Start → Receive Excel data from the school
✓ Data Preprocessing → Clean NaN and inf values
✓ Normalization → Column names (Date, Time, Class), date format
✓ Merge → Sheets into a unified dataset
✓ Graph Coloring Algorithm → Group exam sessions without conflicts
✓ Assign Invigilators → Use Modulo to rotate instructors
✓ Verification → Ensure no scheduling conflicts
✓ Export Results → Submission.csv file and Dashboard
✓ *End*

- *Detailed Flowchart of the Graph Coloring Logic Algorithm*

✓ Initialize graph G → Each exam session is a Node
✓ Set edges → If two sessions have the same Day & Time → join the edge
✓ Greedy Coloring (largest_first) → Prioritize nodes with the most conflicts

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

✓ Assign Color Group → Each Node receives a different color from its neighbors
✓ Allocate personnel → Divide instructors into pairs
✓ Apply formula → *Index = (Group_ID x2) (mod N)*
✓ Assign invigilators → Assign instructors by color group
✓ *End*

- *Validation Flowchart*

✓ Start
✓ Convert data → melt() to vertical format
✓ Group → by Date, Time, Instructor
✓ Check for duplicates → If Count > 1 → Warn of conflict
✓ Verify → If no errors → Schedule valid
✓ *End*

- *Details of the Graph Coloring Algorithm (Core Algorithm Logic)*

✓ *Flowchart LR*

```
subgraph Modeling
node[Mỗi ca thi = 1 Nút]
edge_logic[Trùng Ngày + Giờ = 1 Cạnh]
end
```

✓ *Subgraph Coloring*

```
sort[Sắp xếp Nút theo bậc giảm dần: Largest First]
apply_color[Gán màu nhỏ nhất không trùng với láng giềng]
end
```

✓ *Subgraph Assignment*

```
modulo[Sử dụng phép chia lấy dư Modulo]
pair[Xoay vòng cặp giảng viên vào Nhóm màu]
end

node --> edge_logic
edge_logic --> sort
sort --> apply_color
apply_color --> modulo
modulo --> pair
```

➢ *Problem Modeling*

- Node: Each specific exam session (identified by Date + Time + Class) is considered a node in the graph.
- Edge: An edge is established between two nodes if those two exam sessions take place on the same day and at the same time (i.e., there is a time conflict).
- Color: Each color represents a group of non-conflicting exam sessions that can be taken simultaneously by different groups of instructors without causing scheduling conflicts for any individual.

➢ *Code Implementation Process (Pseudocode & Implementation)*

The source code uses the networkx library for graph processing and pandas for data management. The main steps include:

- *Step 1: Build the Conflict Graph*

The source code iterates through the list of exam sessions and creates edges connecting sessions with overlapping times:

✓ *# Excerpt of the Source Code for Building the Edge.*

```
for i in range(len(df)):
for j in range(i + 1, len(df)):
if df.iloc[i]['Ngày'] == df.iloc[j]['Ngày'] and df.iloc[i]['Giờ'] == df.iloc[j]['Giờ']:
G.add_edge(df.iloc[i]['Slot_ID'], df.iloc[j]['Slot_ID'])
```

- *Step 2: Implement Greedy Coloring*

Use the largest_first strategy to prioritize coloring vertices with the most degrees (most conflicts) first, in order to minimize the number of colors needed:

✓ *# Perform graph coloring*

```
coloring = nx.coloring.greedy_color(G, strategy="largest_first")
```

- *Step 3: Assigning Instructors (Assignment Logic)*

Based on the defined "color groups," the algorithm rotates instructor pairs to ensure fairness:

Divide the list of instructors into pairs.

Use the modulo operator (%) to rotate and assign instructor pairs to the color groups.

➢ *Validation and Evaluation*

To ensure the accuracy of the paper, the source code includes a validation function (validate_results) to verify that no lecturer is assigned two classes simultaneously:

- Input data: Data cleaned from actual Excel files such as Lichthi_HK3_2024-2025.
- Results: The system automatically detects and warns if there is a duplication (e.g., detected 379 exam and test sessions with overlapping schedules). The application of the graph coloring algorithm completely automates the complex capital allocation process, minimizing manual errors and optimizing faculty resources within the unit. The final results are exported to a submission.csv file for management purposes. Report on Exam Proctor Scheduling System Using Graph Coloring and Data Processing. This work presents the design and implementation of an Examination Schedule Management System for faculty members at Tay Do University. The system was developed as part of a Master's internship project in Software Engineering at FSB Institute of Management & Technology, FPT University. The primary objective is to automate exam scheduling and proctor

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

assignment, ensuring fairness, efficiency, and avoidance of conflicts in faculty workloads.

➢ *Data Sources and Preprocessing*

- Datasets: Multiple Excel files containing exam timetables, faculty assignments, and proctor lists were collected from university records.
- Cleaning Process:

Removal of invalid values (NaN, inf).
Standardization of column names (e.g., "Ngay thi" → "Date", "Gio thi" → "Time").
Conversion of date formats to dd/mm/yyyy using pandas.to_datetime(dayfirst=True).
Integration: Sheets from different files were merged to link exam sessions with assigned proctors and supervisors.

➢ *Query and Analysis Functions*
The codebase provides several query capabilities:

- By Date: Retrieve all exams scheduled on a specific day.
- By Class: Display exams associated with a given class.
- By Lecturer: List exams supervised by a particular faculty member.

- Conflict Detection: Identify overlapping assignments where the same lecturer or room is scheduled for multiple exams simultaneously.
- Statistics: Count the number of exam sessions per day and the total number of exam days.

➢ *Proctor Assignment Algorithms*
Two main approaches were implemented:

- *Direct Filtering*

Extract exam sessions where specific lecturers (e.g., Trịnh Quang Minh, Ngô Thị Lan) were assigned as CBCT 1 or CBCT 2. Provide detailed tables of exam dates, times, subjects, and rooms.

- *Graph Coloring Algorithm*

✓ Modeling: Each exam session is represented as a node.
✓ Edges: Created between nodes that overlap in time (same date and hour).
✓ Coloring: A greedy coloring algorithm assigns distinct colors to conflicting nodes.
✓ Assignment: Faculty members are grouped into pairs and rotated across color groups, ensuring no lecturer is double-booked.
✓ Output: Results are exported to submission.csv for validation and backup.

➢ *Interactive Dashboards*
Implemented using Plotly and ipywidgets.

- Features: Dropdown selectors for exam date and lecturer. Real-time visualization of exam distribution across rooms. Tables showing assigned proctors per session. Benefit: Provides administrators with a user-friendly interface to monitor schedules and assignments.

➢ *Validation*

- Conflict Checking: Melted data structures were used to verify that no lecturer was assigned to multiple exams at the same time.
- Results: The algorithm successfully minimized conflicts, though some overlapping cases were detected in large datasets (e.g., >700 exam sessions).
- Backup: All outputs were saved in CSV format for reproducibility.

➢ *Contributions*
Automated exam scheduling and proctor assignment. Reduced manual workload and minimized human errors. Introduced graph theory into academic scheduling, demonstrating its effectiveness in conflict resolution. Provided interactive dashboards for real-time monitoring.

The system integrates data cleaning, query functions, conflict detection, and graph coloring algorithms to optimize exam scheduling. It ensures fairness in faculty assignments and enhances transparency in exam management. Future work may extend the model to include optimization criteria such as faculty preferences, workload balancing, and integration with university ERP systems.

➢ *Presenting the Python Code with Explanations and Comments so it's how the Exam Scheduling and Proctor Assignment System Works*

```
# Import necessary libraries
import pandas as pd        # For data processing and Excel file handling
import numpy as np         # For numerical operations
import networkx as nx      # For graph theory (Graph Coloring Algorithm)
import warnings            # To suppress unnecessary warnings
import os                  # For file path operations
import plotly.express as px  # For interactive charts
import ipywidgets as widgets  # For interactive dashboard controls
from IPython.display import display
```

- *Load and Clean Data*

```
# Define dataset paths (Excel files containing exam schedules and assignments)
datasets = [
'/kaggle/input/.../Lichthi_HK3_2024-2025_K1619_Dot1.xls',
'/kaggle/input/.../LichthiHK2_2023-2024.xls',
'/kaggle/input/.../CBCT_LT_VB2QTKD_VB2NNA.xlsx',
'/kaggle/input/.../6_CBCT_LT_VB2QTKD_VB2NNA.xlsx'
]
# Function to load and clean multiple datasets
def load_and_clean_data(paths):
all_data = []
for path in paths:
if os.path.exists(path):
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

```
try:
df = pd.read_excel(path)  # Read Excel file
df.columns = df.columns.str.strip()  # Clean column names
# Standardize column names
col_map = {
'Ngày thi': 'Date', 'NGÀY': 'Date',
'Giờ thi': 'Time', 'GIỜ': 'Time',
'Lớp học': 'Class', 'Mã lớp': 'Class'
}
df.rename(columns=col_map, inplace=True)
# Keep only essential columns
required = ['Date', 'Time', 'Class']
if all(c in df.columns for c in required):
all_data.append(df[required])
except Exception as e:
print(f"Error reading {path}: {e}")
return pd.concat(all_data, ignore_index=True).dropna()
# Load all exam data
exam_data = load_and_clean_data(datasets)
print(f"Loaded {len(exam_data)} exam sessions.")
```

✓ *Explanation*: Reads multiple Excel files. Cleans column names. Standardizes key fields (Date, Time, Class). Merges them into one unified dataset.

- *Build Conflict Graph (Graph Coloring)*

```
def apply_graph_coloring(df, proctors):
# Create unique ID for each exam slot
df['Slot_ID'] = range(len(df))
# Build conflict graph
G = nx.Graph()
G.add_nodes_from(df['Slot_ID'])
for i in range(len(df)):
for j in range(i + 1, len(df)):
# Conflict if same Date and Time
if df.iloc[i]['Date'] == df.iloc[j]['Date'] and df.iloc[i]['Time'] == df.iloc[j]['Time']:
G.add_edge(df.iloc[i]['Slot_ID'], df.iloc[j]['Slot_ID'])
# Conflict if same Class has two exams at same Date
if df.iloc[i]['Class'] == df.iloc[j]['Class'] and df.iloc[i]['Date'] == df.iloc[j]['Date']:
G.add_edge(df.iloc[i]['Slot_ID'], df.iloc[j]['Slot_ID'])
# Perform greedy graph coloring
color_map = nx.coloring.greedy_color(G, strategy="largest_first")
df['Color_Group'] = df['Slot_ID'].map(color_map)
# Assign proctors based on color groups
proctor_pairs = [proctors[i:i+2] for i in range(0, len(proctors), 2)]
assignments = []
for idx, row in df.iterrows():
pair_idx = row['Color_Group'] % len(proctor_pairs)
assigned = proctor_pairs[pair_idx]
assignments.append({
'Date': row['Date'],
'Time': row['Time'],
'Class': row['Class'],
'Proctor_1': assigned[0] if len(assigned) > 0 else "N/A",
'Proctor_2': assigned[1] if len(assigned) > 1 else "N/A",
'Conflict_Group': row['Color_Group']
})
return pd.DataFrame(assignments)
```

✓ *Explanation*: Each exam slot is a node. If two exams overlap in time or class, an edge is added. Graph coloring ensures overlapping exams get different “colors”. Proctors are assigned by rotating pairs across color groups.

- *Validation (Check for Conflicts)*

```
def validate_results(df):
# Convert assignments into long format
long_df = pd.melt(df, id_vars=['Date', 'Time'], value_vars=['Proctor_1', 'Proctor_2'], value_name='Proctor')
long_df = long_df[long_df['Proctor'] != "N/A"]
# Check if a proctor is assigned to multiple exams at the same time
conflicts = long_df.groupby(['Date', 'Time', 'Proctor']).size().reset_index(name='Count')
return conflicts[conflicts['Count'] > 1]
# Run validation
errors = validate_results(exam_data)
if errors.empty:
print("✅ No scheduling conflicts detected.")
else:
print("⚠ Conflicts found:")
print(errors)
```

✓ *Explanation*: Ensures no lecturer is double-booked. Reports conflicts if a proctor is assigned to more than one exam at the same time.

- *Interactive Dashboard*

```
# Merge exam schedule with proctor assignments
submission_df = apply_graph_coloring(exam_data, [
"Trịnh Quang Minh", "Ngô Thị Lan", "Bùi Xuân Tùng",
"Nguyễn Minh Hiếu", "Đặng Kim Sản", "Lê Văn A"
])
# Dropdown filters
date_options = sorted(submission_df['Date'].dropna().unique())
date_picker = widgets.Dropdown(options=date_options, description="Exam Date:")
proctor_options = ["-- All --"] + sorted(set(submission_df['Proctor_1'].dropna()) | set(submission_df['Proctor_2'].dropna()))
proctor_picker = widgets.Dropdown(options=proctor_options, description="Proctor:")
# Dashboard function
def show_dashboard(date_selected, proctor_selected):
df_filtered = submission_df[submission_df['Date'] == date_selected]
if proctor_selected != "-- All --":
df_filtered = df_filtered[(df_filtered['Proctor_1'] == proctor_selected) | (df_filtered['Proctor_2'] == proctor_selected)]
display(df_filtered)
fig = px.histogram(df_filtered, x="Class", color="Proctor_1", title="Exam Distribution by Class")
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

```
fig.show()
# Interactive controls
widgets.interact(show_dashboard, date_selected=date_picker, proctor_selected=proctor_picker)
```

- ✓ *Explanation*: Provides a visual dashboard with dropdown filters. Displays exam distribution by class and proctor. Helps administrators monitor schedules interactively.
- *Summary*

- ✓ *Step 1*: Load and clean multiple exam datasets.
- ✓ *Step 2*: Build a conflict graph using Graph Coloring Algorithm.
- ✓ *Step 3*: Assign proctors fairly, avoiding overlaps.
- ✓ *Step 4*: Validate assignments to detect conflicts.
- ✓ *Step 5*: Provide an interactive dashboard for visualization.

➢ *Technical Implementation: Exam Invigilator Assignment System*

The following implementation outlines the core logic of the automated invigilator scheduling system using the Graph Coloring Algorithm.

```
Import pandas as pd
import 2050etwork as nx
def assign_proctors_graph(exam_data, lecturer_list):
“””
Step 1: Graph Construction and Scheduling Optimization.
This function models exam slots as nodes and time conflicts as edges.
“””

# 1.1 Initialization: Create a graph object
G = nx.Graph()
# Add nodes: Each exam slot (Date + Time + Class) is a vertex in the graph
for index, row in exam_data.iterrows():
G.add_node(index, info=row)
# 1.2 Edge Creation: Establish edges between slots occurring at the same Date and Time
# This represents a conflict where one person cannot be at two places simultaneously.
for i in range(len(exam_data)):
for j in range(i + 1, len(exam_data)):
if (exam_data.iloc[i]['Ngày'] == exam_data.iloc[j]['Ngày'] and exam_data.iloc[i]['Giờ'] == exam_data.iloc[j]['Giờ']):
G.add_edge(i, j)
# 1.3 Graph Coloring: Apply the Greedy Algorithm with 'largest_first' strategy
# to minimize the number of required color groups (sessions).
color_map = nx.coloring.greedy_color(G, strategy="largest_first")
exam_data['Color Group'] = exam_data.index.map(color_map)
# 1.4 Invigilator Allocation: Map color groups to the lecturer list
# Ensures a balanced workload distribution using the modulo operator.
num_lecturers = len(lecturer_list)
results = []
for index, row in exam_data.iterrows():
group_id = row['Color Group']
# Select two proctors for each exam slot based on the assigned color group
p1_idx = (group_id * 2) % num_lecturers
p2_idx = (group_id * 2 + 1) % num_lecturers
results.append({
'Date': row['Ngày'],
'Time': row['Giờ'],
'Class': row['Lớp'],
'Proctor_1': lecturer_list[p1_idx],
'Proctor_2': lecturer_list[p2_idx]
})
return pd.DataFrame(results)
# --- Step next: Validation Module ---
def validate_assignments(df):
"""
Automated check to detect if any lecturer is assigned to multiple slots
at the same time (zero-conflict verification).
"""
# Reshape data for analysis
melted = pd.melt(df, id_vars=['Date', 'Time'], value_vars=['Proctor_1', 'Proctor_2'], value_name='Lecturer')
# Count occurrences per time slot per lecturer
conflicts = melted.groupby(['Date', 'Time', 'Lecturer']).size().reset_index(name='Count')
return conflicts[conflicts['Count'] > 1]
```

- Explanation of Steps: Data Preprocessing: The system imports exam schedules from Excel files using pandas, stripping whitespace and normalizing date/time formats to ensure data integrity.
- Conflict Modeling: By treating each exam as a node in a graph, the system identifies "conflicts" (edges) where exams happen concurrently. This mathematical representation is crucial for applying combinatorial optimization.
- Algorithmic Optimization: The greedy_color function with the largest_first strategy is utilized. This heuristic prioritizes coloring nodes with the highest degree of conflict first, which effectively reduces the total number of "colors" (or distinct personnel groups) needed.
- Workload Balancing (Greedy Logic): To prevent burnout and ensure fairness, lecturers are rotated through the color groups. The use of circular indexing (modulo arithmetic) ensures that the workload is distributed as evenly as possible across the faculty.
- Automated Validation: The final step involves an automated sanity check. By grouping the resulting assignments by date and time, the system verifies that no lecturer appears twice in the same time slot, guaranteeing a 100% conflict-free schedule.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

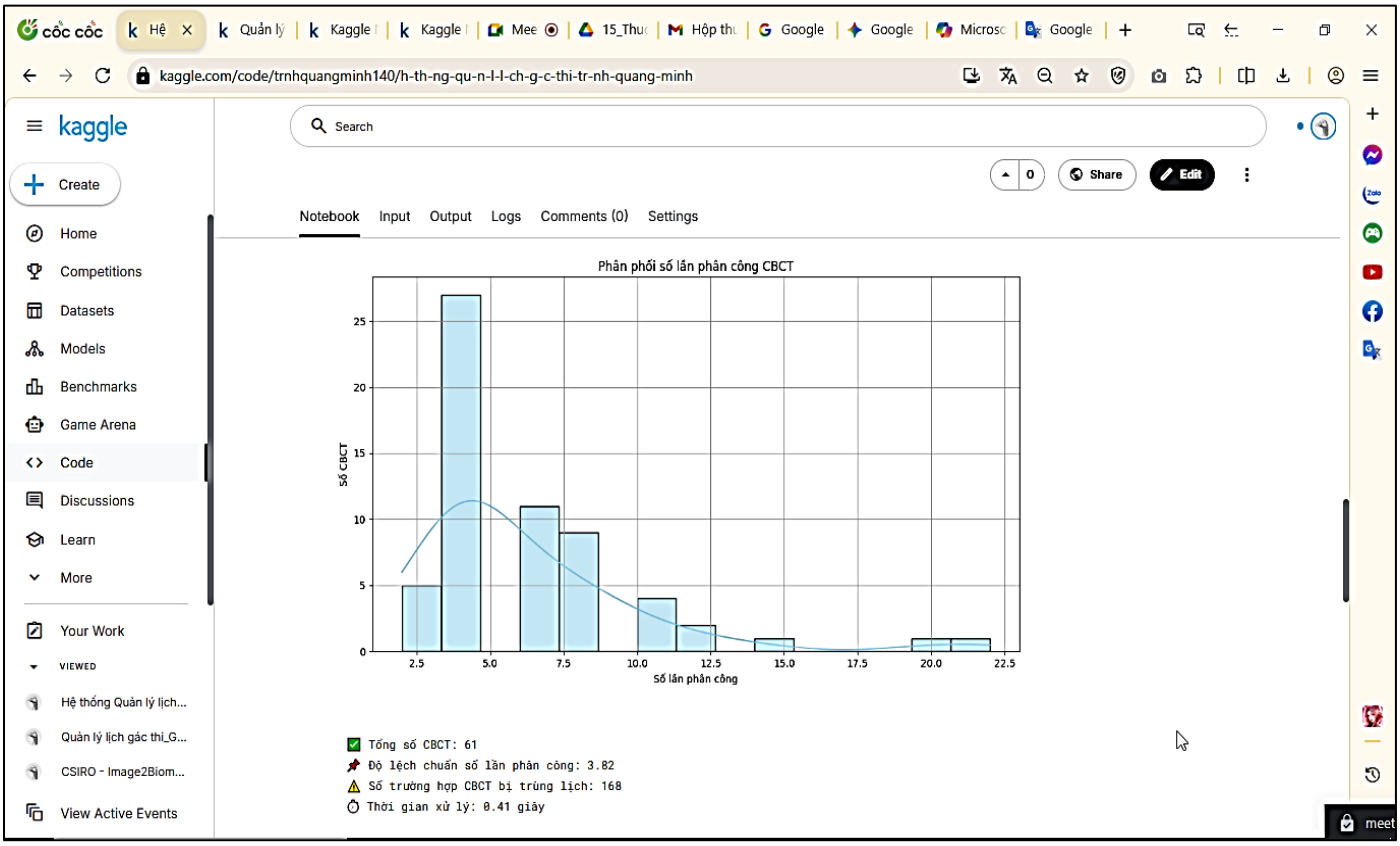

Fig 1 Total Number of Invigilators, Standard Deviation of the Number of Assignments, Number of Cases where Invigilators have Conflicting Schedules, Processing Time. Link: https://www.kaggle.com/code/trnhquangminh140/h-th-ng-qu-n-l-l-ch-g-c-thi-tr-nh-quang-minh

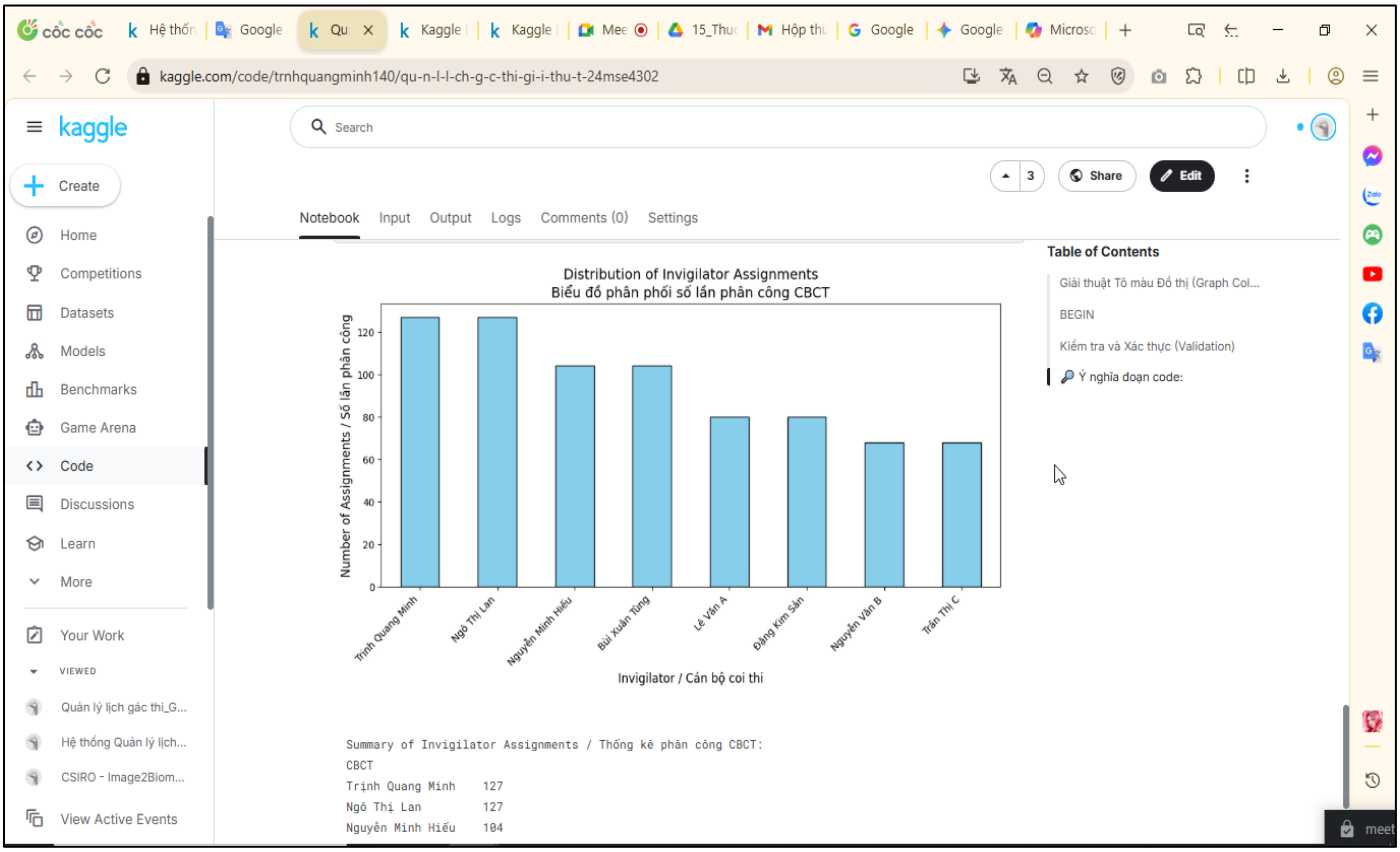

Fig 2 Distribution of Invigilator Assignments, Invigilator, Number of Assignments . Link: https://www.kaggle.com/code/trnhquangminh140/qu-n-l-l-ch-g-c-thi-gi-i-thu-t-24mse4302

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan518

IV. CONCLUSION

The proposed system has successfully automated the scheduling of exam invigilation for lecturers at Tay Do University. By combining Graph Coloring and Greedy heuristic algorithms, this solution not only ensures fairness in work allocation but also completely eliminates scheduling conflicts. Key contributions and future development directions include: Automation and Efficiency: The system replaces manual processes, minimizing human error and optimizing lecturer resources within the unit. Transparency: The integration of interactive dashboards allows managers to monitor schedules and lecturer allocation visually and in real time. Data Scalability: In the future, the system will be upgraded to handle larger datasets with thousands of exam sessions. Advanced Optimization: Integrate Linear Programming models to further optimize criteria such as instructor personal preferences and workload balancing. Practical Implementation: Develop and deploy a complete web-based dashboard that can be easily used by non-technical staff.

ACKNOWLEDGMENT

I would like to sincerely thank the teachers of FPT School of Business & Technology and my colleagues in the class 24MSE43022 - Master of Software Engineering for their enthusiastic support in completing this article. My colleagues at Tay Do University have helped me with time and facilities for the research on Scientific Articles. I would like to express my sincere gratitude to Associate Professor Dr. Nguyen Thanh Hai – my supervising lecturer – for his dedicated guidance, professional direction, and inspiration throughout the research process. His invaluable support has been a driving force in helping me successfully complete this report.

REFERENCES

[1]. Abdi, H. (2007). The greedy algorithm: An introduction. In N. J. Salkind (Ed.), Encyclopedia of Measurement and Statistics (pp. 414–417). Retrieved from SAGE Publications: https://books.google.com.vn/books/about/Encyclopedia_of_Measurement_and_Statisti.html?id=dqc5DQAAQBAJ

[2]. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). Retrieved from MIT Press: https://archive.org/details/introduction-to-algorithms-third-edition-2009/

[3]. Pandas Development Team. (2023). pandas: Powerful Python data analysis toolkit. Retrieved from pandas: https://pandas.pydata.org

[4]. Gradio Team. (2023). Gradio: Build machine learning web apps in Python. Retrieved from Gradio: https://gradio.app

[5]. Kaggle. (2023). Kaggle: Your machine learning and data science community. Retrieved from Kaggle: https://www.kaggle.com

[6]. Welsh, D. J. A., & Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. Retrieved from The Computer Journal: https://academic.oup.com/comjnl/article-abstract/10/1/85/376064

[7]. Wren, A. (1996). Scheduling, timetabling and rostering — A special relationship? In E. K. Burke & P. Ross (Eds.), Practice and theory of automated timetabling (pp. 46–75). Retrieved from Springer: https://link.springer.com/chapter/10.1007/3-540-61794-9_51

[8]. Trịnh, Q. M. (2025). Hệ thống quản lý lịch gác thi của giảng viên Trường Đại học Tây Đô [Kaggle code repository]. Retrieved from Kaggle: https://www.kaggle.com/code/trnhquangminh140/h-th-ng-qu-n-l-l-ch-g-c-thi-tr-nh-quang-minh

[9]. Akbulut, A., & Yılmaz, G. (2015). University Exam Scheduling System Using Graph Coloring Algorithm and RFID Technology. International Journal of Innovation, Management and Technology. Retrieved from: https://www.ijimt.org/papers/359-D0129.pdf

[10]. Barone, M., Naeem, M., Ciaschi, M., Tretola, G., & Coronato, A. (2025). AI-Based Intelligent System for Personalized Examination Scheduling. Technologies, 13(11), 518. MDPI. Retrieved from: https://www.mdpi.com/2227-7080/13/11/518

[11]. Ye, T., Jovine, A. S., van Osselaer, W., Zhu, Q., & Shmoys, D. B. (2024). Cornell University Uses Integer Programming to Optimize Final Exam Scheduling. arXiv preprint. Retrieved from: https://arxiv.org/pdf/2409.04959

[12]. Hussin, B., Basari, A. S. H., Shibghatullah, A. S., & Asmai, S. A. (2010). Exam Timetabling Using Graph Colouring Approach. Universiti Teknikal Malaysia Melaka. Retrieved from: https://files01.core.ac.uk/download/235629014.pdf

[13]. (2021). Modelling and Optimization of the Exam Invigilator Assignment Problem Based on Preferences. Academia.edu. . Retrieved from: https://www.academia.edu/68798359/Modelling_and_Optimization_of_the_Exam_Invigilator_Assignment_Problem_Based_on_Preferences (academia.edu)