Scheduling Algorithms Report

Meet Vyas

April 2023

1 Problem Definition

Create an efficient University Schedule based on the Expression of Interest Data given by the university.

1.1 Introduction

- University course scheduling is the problem of assigning classes to specific times and rooms, such that there are no conflicts and all courses are accommodated in the most efficient manner possible.
- This problem is inherently difficult and has been shown to be NP-Hard.
- In this paper, I propose a novel approach to solve the university course scheduling problem in polynomial time.

1.2 Problem Statement

- The university course scheduling problem is said to be NP-Hard as it falls under the category of scheduling problems.
- Solving a scheduling problem means to find an optimal schedule for a set of activities, subject to constraints.
- The most common scheduling problems are job scheduling, machine scheduling, and project scheduling.
- The university course scheduling problem is a type of job scheduling problem, where each course is a job that needs to be scheduled in a specific time and room.
- The problem is further complicated by the fact that there are multiple courses that need to be scheduled simultaneously, and there are constraints on the availability of rooms and instructors which leads to course clashing and unavailability of several courses to the students.
- The university course scheduling problem is NP-Hard, as it can be reduced to the classic NP-Hard problem of graph coloring.
- Given a graph $G = (V, E)$, the graph coloring problem is to assign a color to each vertex in such a way that no adjacent vertices share the same color.
- This problem is known to be NP-Hard, and can be reduced to the university course scheduling problem as follows:

  1. Create a course for each vertex in the graph G.
  2. If there is an edge between two vertices u and v in G, create a constraint that the courses corresponding to u and v cannot be scheduled at the same time and room.

Thus, we have shown that the university course scheduling problem is NP-Hard, and requires an efficient algorithm to solve it.

2 Reference Material

Scheduling Algorithms by Doctor Peter Brucker [1]

3 Method of Approach

- I started by reading the textbook's first two chapters, which comprised the following information mentioned in the below sub sections.
- The book talks about the fundamentals of scheduling algorithms which form the core of the subject operations research. These algorithms help optimise resource utilisation, reduce costs, and improve efficiency in various fields like manufacturing, transportation, healthcare, and telecommunication.
- The first chapter discusses the different use cases of scheduling algorithms in modern industries. These algorithms are used to solve complex problems such as routing, sequencing and allocating resources in a time dependent matter.
- The chapter further discusses the different scheduling problems such as single machine scheduling, parallel machine scheduling, flow shop scheduling, job shop scheduling, and open shop scheduling.

3.1 Scheduling Variables

[1]

Job Data

- The first chapter goes in depth about the classification of problems and the various variables used in the process. The chapter starts with the explanation of Job Data which consists of a number n_i of operations. The job data with be denoted as $O_{i1}, O_{i2}...O_{in_i}$.

Processing Requirement

- If J_i contains $n_i = 1$, then we identify J_i with O_{i_1} and denote the processing requirement as P_i. Furthermore a release date r_i, on which operation of J_i for the first time becomes available for processing can be specified.
- Associated with each operation O_ij is a set of machines $\mu_ij \subseteq \{M_1, M_2..., M_m\}$
- O_{ij} may be processed in any of the machines in μ_{ij}.
- All μ_{ij} are one element sets or all μ_{ij} are equal to set of all machines.
- In the first case we have dedicated machines and in second case we have parallel machines.
- Machines are equipped with different tools which covers problems in Flexible Manufacturing, these machines are known as multipurpose machines.
- One more possibility is that machines in the set μ_{ij} are used simultaneously by O_{ij} during the entire processing period.
- Scheduling Problems like these are called multiprocessor task scheduling problems.
- The last data consists of a cost function $f_i(t)$ which measures the cost of completing J_i at time t. A due date d_i and a weight w_i can be used in defining f_i.
- In general, all data $p_i, p_{ij}, r_i, di, w_i$ are assumed to be integer.
- A schedule is feasible if no two time intervals overlap on the same machine, if no two time intervals allocated to the same job overlap, and if, in addition, it meets a number of problem-specific characteristics.
- A schedule is optimal if it minimizes a given optimality criterion(γ).
- Classes if scheduling problems are specified in terms of a three field classification $\alpha|\beta|\gamma$ where α specifies the machine environment, β specifies Job Characteristics.

Job Characteristics (β)

- They are a set called β containing at most six elements $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$ and β_6.
- β_1 indicates whether preemption(or job splitting) is allowed. Preemption means that processing may be interrupted and resumed at a later time, even on another machine. A job or operation may be interrupted several times. If allowed we set $\beta_1 = pmtn$. Otherwise β_1 does not appear in β.
- β_2 describes precedence relations between jobs. These precedence relations may be represented by an acyclic directed graph $(G = V, A)$ where V = $\{1, 2, 3..., n\}$ corresponds with jobs, and $(i, k) \in A$ if J_i must be completed before J_k starts. In this case we write $J_i \rightarrow J_k$.

- If G is an arbitrary acyclic directed graph we set $\beta_2 =$ **prec**. Sometimes we will consider scheduling problems with restricted precedences given by chains, intree, outtree, and sp-graph. If $\beta_2 =$ intree, then G is a rooted tree with an outdegree for each vertex of at the most one. If $\beta_2 =$ outtree, then G is a rooted tree with an indegree for each vertex of at most one. Thus in an intree all arcs are directed towards a root and in an outtree all arcs are directed towards away from a root. If $\beta_2 =$ tree, then G is either an intree or an outtree. A set of chains is a tree in which the outdegree and indegree for each vertex is at the most one. If $\beta_2 =$ chains, then G is a set of chains.
- Series-parallel graphs are closely related to trees.
- A graph is called series-parallel if it can be built by means of the following rules:
  - Base Graph: Any graph consisting of a single vertex is series-parallel. Let $G_i = (V_i, A_i)$ be series-parallel $(i = 1, 2)$.
  - Parallel composition: The graph $G = (V1 \cup V2, A1 \cup A2)$ formed from G1 and G2 by joining the vertex sets and arc sets is series parallel.
  - Series composition. The graph $G = (V1 \cup V2, A1 \cup A2 \cup T1 \times S2)$formed from G1 and G2 by joining the vertex sets and arc sets and adding all arcs (t, s) where t belongs to the set T1 of sinks of G1 (i.e. the set of vertices without successors) and s belongs to the set S2 of sources of G2 (i.e. the set of vertices without predecessors) is series parallel.
- We set $\beta_2 =$ sp-graph if G is series parallel. If there are no precedence constraints, then β_2 does not appear in β.
- If $\beta_3 = r_i$, then release dates may be specified for each job. If $r_i = 0$ for all jobs, then $beta_3$ does not appear in $beta$.
- β_4 specifies restrictions on the processing times or on the number of operations. If $beta_4$ is equal to $p_i = 1(p_{ij} = 1)$, then each job (operation) has a unit processing requirement. Similarly, we may write $p_i = p(p_{ij} = p)$. Occasionally, the $beta_4$ field contains additional characteristics with an obvious interpretation such as $pi \in \{1, 2\}$ or $d_i = d$.
- If $\beta_5 = d_1$, then a deadline d_i, is specified for each job J_i, i.e job J_i must not finish later than time d_i.
- In some scheduling applications, sets of jobs must be grouped into batches.
- A batch is a set of jobs which must be processed jointly on a machine. The finishing time of all jobs in a batch is defined as equal to the finishing time of the batch.
- A batch may consist of a single job up to n jobs. There is a set-up time s for each batch.
- We assume that this set-up time is the same for all batches and sequence independent.
- A batching problem is to group the jobs into batches and to schedule these batches. There are two types of batching problems, denoted by p-batching problems, and s-batching problems.
- For p-batching problems (s batching-problems) the length of a batch is equal to the maximum(sum) of processing times of all jobs in the batch.
- $\beta_6 =$ p-batch or $\beta_6 =$ s-batch indicates a batching problem. Otherwise β_6 does not appear in β.

Machine Environment

- The machine environment is characterized by a string $\alpha = \alpha_1\alpha_2$ of two parameters. Possible values of α_1 are $\circ, P, Q, R, PMPM, QMPM, G, X, O, J, F$.
- If $\alpha_1 \in \circ, P, Q, R, PMPM, QMPM$, where $\circ$ denotes the empty symbol (thus, $\alpha = \alpha_2$ if $\alpha_1 = \circ$), then each J_i consists of a single operation.
- If $\alpha_1 = \circ$, each job must be processed on a specified dedicated machine.
- If $\alpha_1 \in \{P, Q, R\}$, then we have parallel machines, i.e. each job can be processed on each of the machines $M_1, \ldots, M_m$. If $\alpha_1 = P$, then there are identical parallel machines. Thus, for the processing time p_{ij} of job J_i on M_j we have $p_{ij} = p_i$ for all machines M_j. If $\alpha_1 = Q$, then there are uniform parallel machines, i.e. $p_{ij} = p_i/s_j$ where s_j is the speed of machine M_j. Finally, if $\alpha_1 = R$, then there are unrelated parallel machines, i.e. $p_{ij} = p_i/s_{ij}$ for job-dependent speeds s_{ij} of M_j.
- If $\alpha_1 = PMPM$ and $\alpha_1 = QMPM$, then we have multi-purpose machines with identical and uniform speeds, respectively.

– If $\alpha_1 \in \{G, X, O, J, F\}$, we have a multi-operation model, i.e. associated with each job J_i there is a set of operations $O_{i1}, \ldots, O_{in_i}$. The machines are dedicated, i.e. all μ_{ij} are one element sets. Furthermore, there are precedence relations between arbitrary operations. This general model is called a general shop. We indicate the general shop by setting $\alpha_1 = G$. Job shops, flow shops, open shops, and mixed shops are special cases of the general shop. In a job shop, indicated by $\alpha_1 = J$, we have special precedence relations of the form

$$O_{i1} \to O_{i2} \to O_{i3} \to \cdots \to O_{in_i} \quad \text{for } i = 1, \ldots, n.$$

Furthermore, we generally assume that $\mu_{ij} \neq \mu_{i,j+1}$ for $j = 1, \ldots, n_i - 1$. We call a job shop in which $\mu_{ij} = \mu_{i,j+1}$ is possible a job shop with machine repetition.

– The flow shop, indicated by $\alpha_1 = F$, is a special case of the job-shop in which $n_i = m$ for $i = 1, \ldots, n$ and $\mu_{ij} = \{M_j\}$ for each $i = 1, \ldots, n$ and $\mu_{ij}\& = M_j$ for each i = 1, . . . , n and j = 1, . . . , m.

– The open shop, denoted by $\alpha_1 = O$, is defined as the flow shop, with the exception that there are no precedence relations between the operations. A mixed shop, indicated by $\alpha_1 = X$, is a combination of a job shop and an open shop.

– A permutation flow shop is a flow shop in which jobs are processed in the same order on each machine. Figure 1.2 shows a feasible schedule for a permutation flow shop. If we have a job shop problem, we may set β_4 equal to $n_i \leq 2$. In this case, all jobs have at most two operations. If α_2 is equal to a positive integer $1, 2, \ldots$, then α_2 denotes the number of machines. If $\alpha_2 = k$, then k is an arbitrary but fixed number of machines. If the number of machines is arbitrary, we set $\alpha_2 = \circ$

4 Using Gephi for Graph Visualisation

In this report, I present the process and results of visualizing a university course schedule using the Gephi software. Gephi is an open-source network analysis and visualization software package that allows users to explore and analyze complex networks. I used the Fruchterman-Reingold algorithm for graph layout, which is a force-directed algorithm that simulates a physical system of nodes connected by edges.

4.1 Data Preparation

– I obtained the course schedule data in the form of an edge list by running a python code which creates a source target list from the Expression of Interest Data for the Winter Semester 2022, which is a set of connections between nodes.

– Each node represents a course, and the edges represent the relationships between the courses.

– I prepared the data by importing it into Gephi using import spreadsheet and adjusting the settings for visualization.

4.2 Visualization with Gephi

– Gephi provides a user-friendly interface for creating and manipulating graphs along with various statistical tools. I used the Fruchterman-Reingold algorithm to layout the nodes and edges. This algorithm simulates a physical system of nodes connected by edges, where nodes repel each other and edges attract the nodes they connect.

– I set the gravity parameter to 10000, which controls the strength of the attraction between the edges and nodes. This high value creates a tight layout with closely connected nodes, which helps to highlight the relationships between courses.

4.3 Results

– The resulting graph provided a visual representation of the course schedule that allows for easy identification of the relationships between courses and their clustering. I observed that courses in related fields tend to cluster together, indicating that they have similar requirements or are part of the same program. I also observed that some courses act as bridges between different clusters, indicating that they may be prerequisites for courses in multiple fields.

– Overall, the graph helped in understanding the course schedule and identifying potential areas for improvement or optimization as the graph was very clustered. I was then given the advice to work with Graph Colouring due the nature of the graph. The Gephi software and Fruchterman-Reingold algorithm were essential tools in creating this visualization, moving further into the project and analysis.

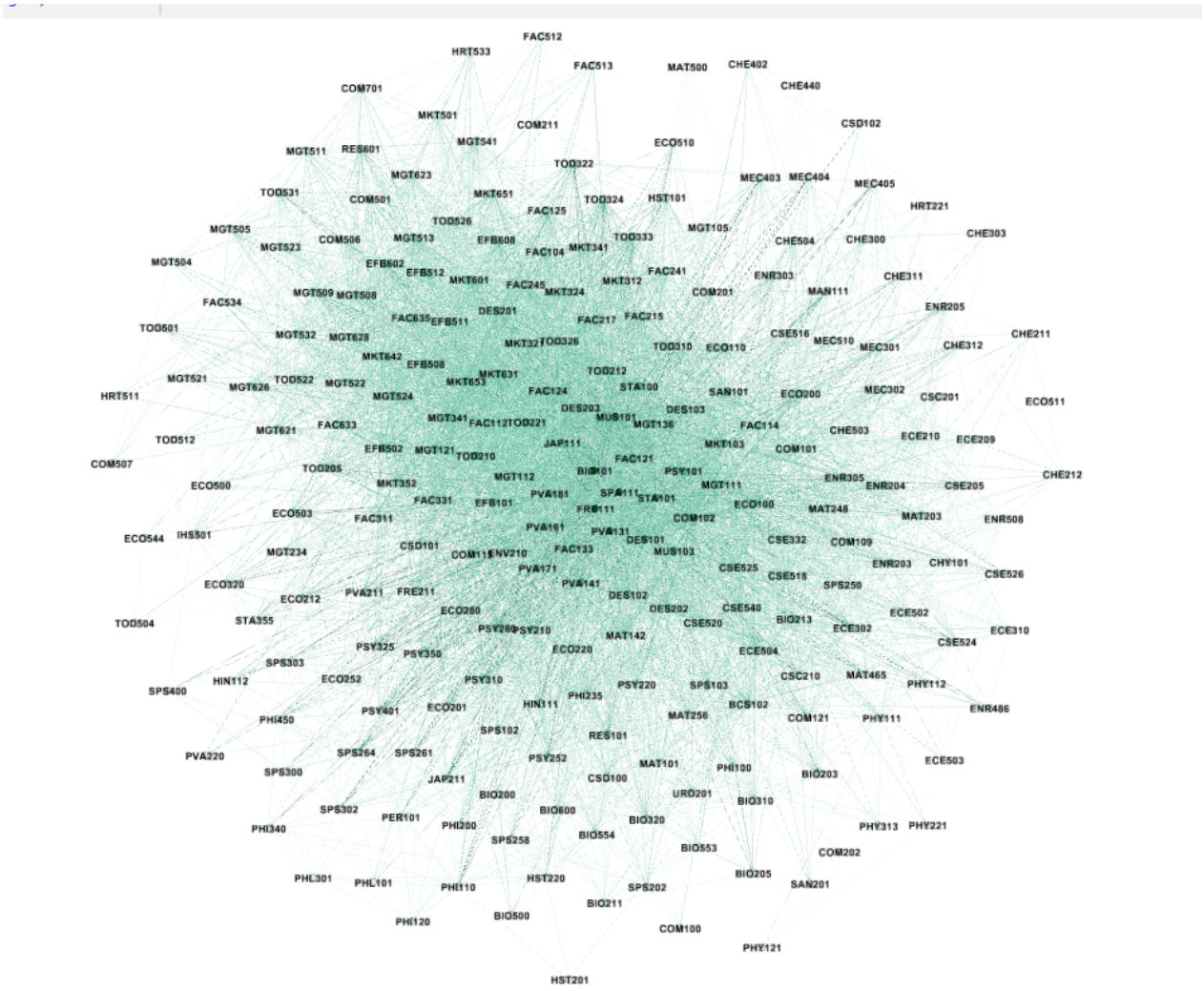

Figure 1: Gephi Graph

5 NetworkX

NetworkX is a Python package used for creating, manipulating, and analyzing complex networks and graphs.

5.1 NetworkX as a Graph Library in Python

– NetworkX is a library in Python that provides tools for creating, manipulating, and analyzing complex networks and graphs.

– It is built on top of the numerical library NumPy and offers a high level of functionality for constructing graphs, manipulating graph structures, and analyzing network properties.

– The library provides an interface for handling graphs and has support for directed, undirected, weighted, and unweighted graphs.

5.2 Partitioning Graphs into Communities

- Graph partitioning is a fundamental problem in graph theory that involves dividing a graph into smaller subgraphs or communities based on some criteria. NetworkX provides several algorithms for partitioning graphs into communities, including the Louvain method and the Girvan-Newman method out of which I have used the Louvian method.
- The Louvain method is a hierarchical clustering algorithm that optimizes modularity, a measure of the quality of the partitioning. It is used for detecting communities in large networks and was used to partition the university edge list graph into communities.

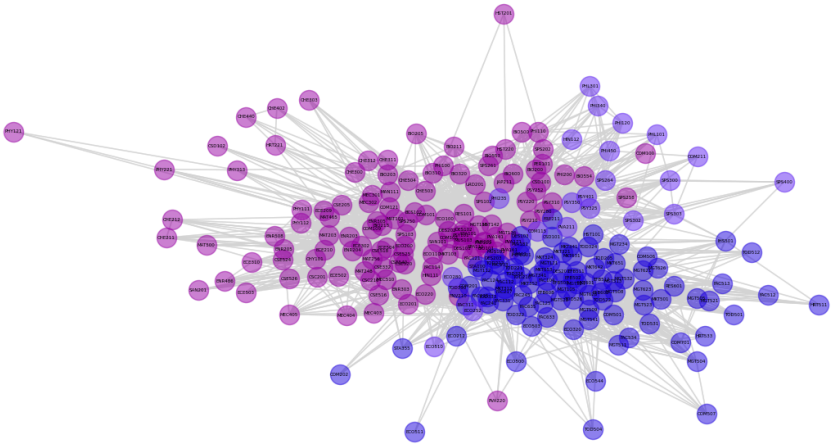

Figure 2: Graph Partitioned with Louvain Communities

5.3 Graph Colouring

- Graph coloring is another fundamental problem in graph theory that involves assigning colors to the vertices of a graph such that no two adjacent vertices have the same color. NetworkX provides a range of algorithms for graph coloring, where I have used greedy coloring,
- The greedy coloring algorithm is a simple algorithm that starts with an empty color set and assigns colors to the vertices one by one, choosing the color that has been used the least by its neighbors. This algorithm has a time complexity of $O(n^2)$ where n is the number of vertices in the graph.
- Graph coloring is a useful tool for a variety of applications, such as scheduling, map coloring, and frequency assignment. In the context of university course scheduling, graph coloring can be used to assign time slots to courses such that no two courses with overlapping schedules are assigned the same time slot. This can help ensure that students can take all the courses they need without scheduling conflicts.

6 Solution

6.1 Polynomial Time Solution

The solution I reached is a novel approach to solve the university course scheduling in polynomial time. The approach involved using community detection algorithms to partition the graph into subgraphs, and then using graph coloring algorithms to assign courses to specific time and room slots.

Specifically, I used the Louvain community detection algorithm to partition the graph into subgraphs. The Louvain algorithm is a popular community detection algorithm that optimizes modularity, which measures the degree to

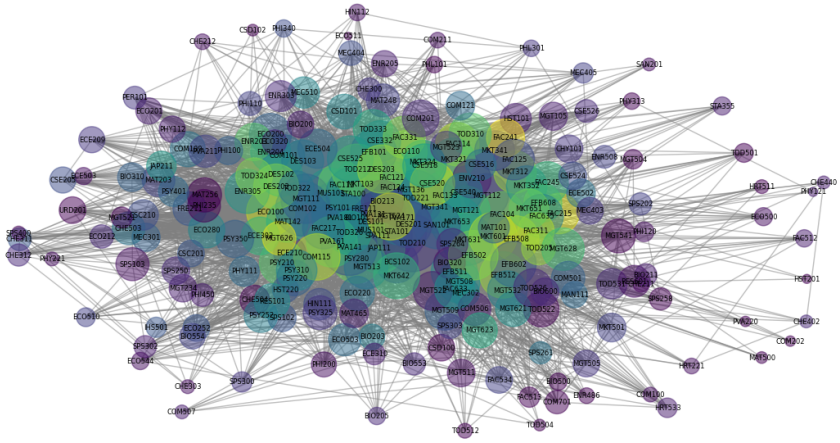

Figure 3: Coloured graph without partitioning

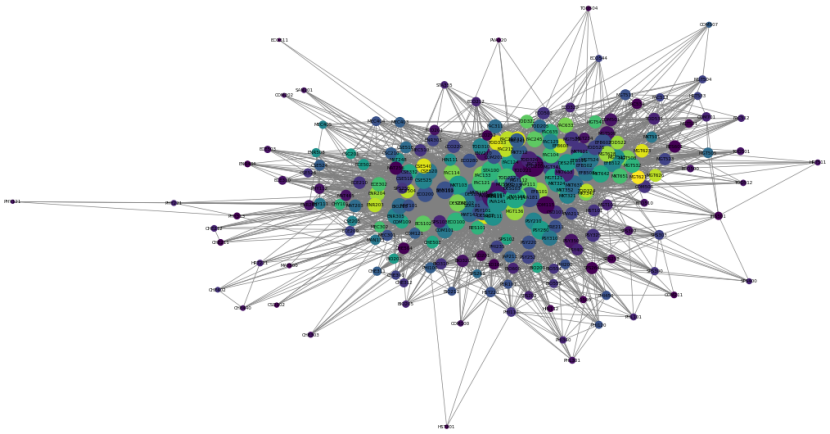

Figure 4: Coloured graph after partitioning

which a graph can be partitioned into non-overlapping communities. The algorithm is iterative, and involves merging and splitting communities to maximize modularity.

Once the graph has been partitioned into subgraphs, I used the greedy coloring algorithm to assign courses to specific time and room slots. The greedy coloring algorithm is a simple and efficient algorithm that assigns colors to vertices in a sequential manner. At each step, the algorithm assigns the lowest available color to the current vertex, and then updates the set of available colors for the adjacent vertices.

The advantage of our approach is that it can solve the university course scheduling problem in polynomial time, which reaches the goal of the project. Furthermore, our approach is scalable and can handle large graphs with thousands of vertices.

7 Google OR Tools

- Google OR Tools was another approach used to solve the university scheduling problem.
- The Google OR Tools is a suite created by google to solve complex problems in combinatorial optimization problems.
- For the problem of university course scheduling, we can model the problem as a constraint satisfaction problem and use the package and it's built in solvers to solve the problem. The OR-Tools solver is based on the CP-SAT algorithm, which is a hybrid of constraint programming and linear programming.
- Using the nurse scheduling problem of the Employee Scheduling section as a reference for solving the problem, attempts were made to create a viable schedule using different objective functions.
- The different objective functions include maximizing the students assigned to courses, which would be the scenario where the maximum enrollment or maximum utilization of available course capacity is done to reduce clashes between courses. Another objective function tried out was to reduce the number of course clashes between students.
- The approach didn't work for the time being as the CPSAT Solver took too much computation time and resources and a clear schedule could not be reached that minimized the number of course clashes.

8 Exam Scheduling

Exam Scheduling can also be done using the same way as course scheduling only the parameters for the days and slots need to be changed, as exams can be scheduled to a maximum of three per day for fairness.

9 Solution variations changing the methods and resolution of Louvian Methods in Graph Partitioning

- Without partitioning the graph a schedule was created which did not have any clashes in it, which is a proof of concept that the graph colouring algorithm works and a viable schedule is obtained.
- With partitioning, we create communities by modularity, in which small communities are found by optimizing modularity locally on all nodes. Then, graph coloring is applied on each community individual, thus balancing a trade-off in minimizing clashes and time slots.
- The Louvain community detection algorithm is a popular method for identifying communities or clusters within a network. The algorithm aims to maximize the modularity of the network, which is a measure of the density of connections within communities compared to the expected density of connections in a random network.
- The modularity of a network is defined as:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j), \tag{1}$$

  where A_{ij} is the weight of the edge between nodes i and j, k_i and k_j are the sums of the weights of the edges connected to nodes i and j, respectively, m is the total weight of the edges in the network, c_i is the community to which node i belongs, and $\delta(c_i, c_j)$ is the Kronecker delta function, which equals 1 if nodes i and j belong to the same community and 0 otherwise.
- The Louvain algorithm works by iteratively optimizing modularity at the local level. In the first phase, each node is assigned to its own community, and the algorithm tries to improve modularity by moving nodes between communities. This is done by considering the change in modularity that would result from moving a node to its neighbor's community, and selecting the move that results in the largest increase in modularity. This process is repeated until no further improvement in modularity can be achieved.
- In the second phase, the communities identified in the first phase are treated as nodes in a new network, and the same process is repeated to identify communities at a larger scale.
- On a node level, this algorithm thus maximizes the edge weights between this node and the community it belongs to, and minimizes the edge weights crossing into other communities. On a global level, the algorithm maximizes intra-community (within community) edge weights, while minimizing the inter-community (between communities) edge crossing weights.

– This allows us to find communities where there will be a higher tendency of clashes, whereas between communities, there will be a lower level of clashes for scheduling. We can then color all of these communities independently, to achieve a balanced trade-off between minimizing clashes and also minimizing node colors required (which are the required time slots).

9.1 Psuedocode

```
Read in edge list from a CSV file named 'test.csv'
Create an empty list named 'edgelist'
Loop through each row in the CSV file using csv.reader
Append the first and second elements of each row to the 'edgelist' as a tuple
Measure the time taken for reading in the edge list using the 'timeit.default_timer()' function and store the start time
Create a graph 'G' from the 'edgelist' using networkx.Graph()
Partition the graph 'G' into communities using the Louvain algorithm with a resolution of 5
Store the partitioned communities in a variable named 'communities'
Print the 'communities'
Measure the time taken for partitioning into communities using the 'timeit.default_timer()' function and subtract the start time to get the time taken, store it in 'partition_time'
Create an empty list named 'schedules'
For each community in the 'communities':
Create a subgraph of the community
Assign colors to each node in the subgraph using the greedy coloring algorithm with 'saturation_largest_first' strategy
Store each course and its corresponding time slot in a dictionary 'schedule'
Append each 'schedule' to the 'schedules'
Measure the time taken for assigning time slots using the 'timeit.default_timer()' function and subtract the 'partition_time' to get the time taken, store it in 'assignment_time'
Create an empty pandas DataFrame named 'schedule_matrix' with columns labeled 'Day 1' to 'Day 5'
Create an empty set named 'scheduled_courses'
Measure the time taken for creating the 'schedule_matrix' using the 'timeit.default_timer()' function and store the start time
For each time slot in the range of 1 to 15:
Calculate the day number of the time slot using the formula (slot - 1) % 5 + 1
Create an empty row in the 'schedule_matrix' for the current time slot
For each community in 'schedules':
Get all courses scheduled for the current time slot and the current community
For each course, if it has not been scheduled before, add it to the current row of the 'schedule_matrix' for the current day
Add all scheduled courses for the current time slot to the 'scheduled_courses' set
Measure the time taken for populating the 'schedule_matrix' using the 'timeit.default_timer()' function and subtract the start time to get the time taken, store it in 'schedule_matrix_time'
Save the 'schedule_matrix' to an Excel file named 'university_schedule1.xlsx'
Measure the time taken for saving the 'schedule_matrix' to Excel using the 'timeit.default_timer()' function and subtract the 'schedule_matrix_time' to get the time taken, store it in 'saving_time'
Calculate the total time taken by adding 'partition_time', 'assignment_time', 'schedule_matrix_time', and 'saving_time', and store it in 'total_time'
Print the time taken for each step and the total time taken using formatted strings.
```

10 Future Prospects

Currently the schedule created only schedules all the courses once per 15 slots for the course scheduling and 3 exams per day for 7 days for exam scheduling. A lot of work can be done to further enhance the understanding of the course and reach a truly novel and optimum solution for the University Course Scheduling Problem.

10.1 Changing the current approach

– Different algorithms for graph colouring can be used to get the best result for the graph colouring problem. As the graph coloring problem is known to be NP-complete there is no known algorithm which, for every graph, will optimally color the nodes of the graph in a time bounded by a polynomial in the number of nodes.

– Many graph coloring algorithm such as the Saturation algorithm, the Recursive Largest First algorithm, Simulated Annealing algorithm, Greedy algorithm, are NP complete. I have used the greedy algorithm in the

solution, but other algorithms can be tested such as the largest first or degree of saturation problem. or a new algorithm can be developed which gives a better solution to the problem.

- Along with graph colouring, different algorithms can be applied to the same problem. In my solution I have used Louvian communities as well as greedy modularity communities which are both NP complete algorithms and thus we have to use a heuristic approach for the same.Here we have set the resolution for both the algorithms to 5 which results in smaller communities.
- Other approaches to the problem such as genetic algorithms and simulated annealing can also be tried out to better the chances of an optimum solution for the university schedule problem.

10.2 Updating and advancements in the current approach

- A new edge list can be created with the weights of the nodes which would help in better allocation of the courses to different communities and colouring algorithms.
- Different courses with different credits can be allocated in a schedule in such a way that two credit courses have a certain time of running and certain slots they can be run in. We can differentiate these constraints in two categories as hard constraints and soft constraints. [2]
- Lab courses and courses with different credits can have different constraints which need to be satisfied and thus will fall into the category of hard constraints.
- Faculty time preferences, room allocation, batch capacity preferences don't strictly need to be satisfied and thus will fall under the category of soft constraints.
- So the idea can be formalized further and a better solution to the problem can be obtained.

11 Comparision of Distance/Similarity Based Approaches

Methodology	**No. of Clashes**
node2vec	1450
LouvainNE	3675
RandomProjection	2810
Singular Value Decomposition	4665
Generalised Singular Value Decomposition	2710
Spectral	4323
Principal Component Analysis	5967
LouvainEmbedding	2931
Louvain Recursive Patition Tree	7170

Note: all embeddings with dimensionality of 5 and L1 loss.
No. of simultaneous slots/rooms fixed to 10.

Table 1: Number of clashes for different embedding approaches

12 Conclusion

Thus I reached the goal of creating a simple schedule with a scheduling algorithm and I understood the implementation and methods of the major algorithms used for course scheduling.

References

[1] Brucker, P. (2007). Scheduling algorithms. Springer.

[2] Ganguli, R., & Roy, S. (2017). A study on course timetable scheduling using graph coloring approach. *International Journal of Computational and Applied Mathematics*, *12*(2), 469-485.

[3] San, S., Ong, K., See, J., & Abdullah, L. (2012). A university timetabling system based on graph colouring and constraint manipulation. *Applied Mathematical Sciences*, 6(38), 1883-1892.

[4] Kwok, J., & Leung, H. (2007). Graph coloring for class scheduling. *ACM Sigada Ada Letters*, 27(3), 27-33.

[5] Hacıoğlu, Y., & Yilmaz, I. (2008). Course timetabling by graph coloring. *The Engineering and Architecture Faculty of Gazi University Journal*, 23(4), 791-799.

[6] Islam, M. N., Giri, M. A. S., & Farhana, S. (2017). Comparative analysis of graph coloring algorithms for scheduling problem. *International Journal of Computer Applications in Management*, 12(2), 25-33.

[7] Ibraheem, A., & Salim, N. H. (2017). Solving the class scheduling problem using graph coloring technique. *International Journal of Applied Engineering Research*, 12(21), 10499-10510.

[8] Google Optimization. (n.d.). Retrieved April 24, 2023, from https://developers.google.com/optimization

[9] Hagberg, Aric, Swart, Pieter, & Chultunbaev, Anton. (2021). NetworkX Documentation. Retrieved April 24, 2023, from https://networkx.org/documentation/stable/.

[10] Louvain method. (2023, March 22). In Wikipedia. Retrieved April 24, 2023, from https://en.wikipedia.org/wiki/Louvainmethod Algorithm