Solving Exam Scheduling Problem Using Graph Coloring

A Final Report

Presented to

Professor Katerina Potika

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 255

By

Fei Pan,

Joshua Benz

30 April, 2019

TABLE OF CONTENTS

I. INTRODUCTION

Graphs are useful data structures for modeling relationships between different types of data. The applications of graphs include modelling social media interactions , modelling maps for path optimizations, recommendation engines and more. Once a graph is constructed, there are a variety of algorithms that can be used to obtain information from them. In this project, we examine a special case known as graph coloring.

The graph coloring problem is to assign colors to every vertex in theo graph such that no adjacent vertices have the same color assignment. The goal of this problem is to minimize the number of colors needed to color the graph. Graph coloring is NP-Complete which means there is no polynomial time algorithm for finding the optimal solution. Therefore, we explore approximations.

This project examines a Greedy approach, Welsh-Powell and a Genetic algorithm approach to obtaining a valid graph coloring for undirected graphs. All of these algorithms are approximations and do not guarantee optimal solutions. In the case of the genetic algorithm, it does not guarantee a solution at all. We will apply these algorithms to graphs that model university final exam scheduling.

II. DEFINITION OF THE PROBLEM

A graph is said to be colored if a color has been assigned to each vertex in such a way that adjacent vertices have different colors. And this process of assigning colors to the vertices of a graph is called graph coloring. Suppose we want to schedule some exams for SJSU students. The problem can be converted into a graph coloring problem. Given a graph G(V, E), in which the vertices V represent courses and two vertices are connected as an edge E if the corresponding courses have a student in common. Two connected vertices should be colored differently. As in exam scheduling, no student will be scheduled to take two course exams simultaneously. The objective of this problem is to find a complete solution that every course is assigned to a time slot and a room, and to minimize the total number of time periods, which is represented as the chromatic number of a graph.

The objective of this problem is to find a complete solution that every course is assigned to a time slot and a room, and to minimize the total number of time periods / chromatic number of a graph.

III. ALGORITHM DESCRIPTIONS

Greedy

1. Select the first vertex and make it the first color

2. For each vertex in the graph

(a) Visit each neighbor and take note of the colors assigned to the neighbors

(b) Pick the first color that isn't assigned to a neighbor

A. Welsh-Powell

1. Find the degree of each vertex
2. List the vertices in order of descending degrees.
3. Color the first vertex with color 1.
4. Move down the list and color all the vertices not connected to the colored vertex, with the same color.
5. Repeat step (1) on all uncolored vertices with a new color, in descending order of degrees until all the vertices are coloured.

B. Genetic Algorithm

1. Create an initial population of genomes (randomly or not)
2. While Fitness is not 0 and we have not reached 20,000 iterations
   (a) For each genome, do the following
      i. Update its fitness score based on the number of bad edges
      ii. If we have an optimal fitness score, stop the program
   (b) Select two parents and do the following
      i. Crossover the parents to make a new child
   (c) Mutate the child
   (d) Place the child in the population
   (e) Place the fitter of the two parents into the population (this wasn't specified, it's just how I decided to do it)
   (f) If we reach 20,000 iteration, stop and perform wisdomOfTheArtificialCrowds

In this algorithm, a genome holds a chromosome and an associated fitness score. The chromosome is an array based representation of the coloring of the graph. The position in the array represents the vertex in the graph while the value at the position is the assigned color for that vertex. "Bad edges" are defined as edges between vertices of the same color. Parent selection is done by either selecting the top two performing

genomes in the population or by randomly selecting two genomes from the population. Mutation is done by looking at a chromosome and looking and taking each position as a vertex. We get the neighbors of the vertex and keep track of what colors are used. At the end, the vertex gets assigned a random color out of the difference of AllColors - usedColors. In other words, pick a random color that isn't used by any of the neighbors. WisdomOfTheArtificialCrowds is performed by taking the top performing chromosome and looking at its bad edges. Then you take the top 50% of performers and take a majority vote on what color they use that works. Then we assign that color for the vertex in the top performer. The size of the population, mutation rate and number of iterations were experimentally chosen.

IV. IMPLEMENTATION DETAILS

The above algorithms were implemented in python with use of the pandas library.

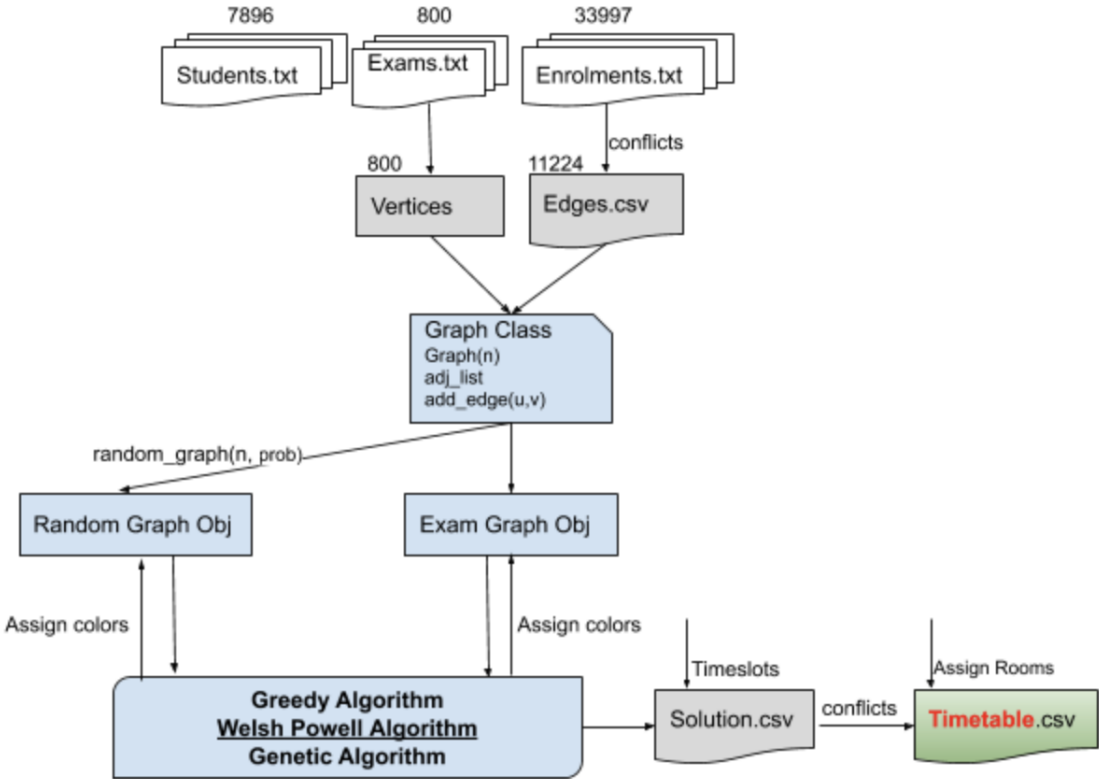

Figure 1: Execution Flow

V. RELATED WORK

Graph coloring is an NP-Complete problem, so there has been a lot of work done to find better algorithms and special cases. In this project, we implemented the standard Greedy Algorithm, Welsh-Powell and Genetic algorithm. .....NUMBER

VI. DATA SETS USED

1. This data is from the University of Nottingham, semester 1, 1994-95. There are 800 exams provided by 46 departments, and 7896 students. There are 33997 enrollments, so each student is sitting an average of 4.3 exams, and each exam is sat by an average of 42.5 students.

2. Graphs randomly generated by following the Erdős–Rényi model. This algorithm works by selecting the number of vertices and a probability of them being connected. Then I generated a random number and if it was less than or equal to the probability, then I added an edge between the vertices.

VII. ANALYSIS AND COMPARISON

Graph	Dimensions	**Greedy**	**Welsh-Powell**	**Genetic**
Random Graph	($\|V\| = 800$, $\|E\| = 12716$)	15	13	-
Exam Graph	($\|V\| = 800$, $\|E\| = 11224$)	26	19	39

VIII. EXPERIMENTAL RESULTS

IX. FURTHER DISCUSSION

A. Genetic Algorithm

A..1 Implementation

The Genetic Algorithm essentially tried to apply a modified version of the Greedy algorithm at random starting points and continually building a better graph over the generations. The initial population starts at random and when the top performing parents are selected, the child inherits their good and bad choices. This is call the crossover stage. At the next stage, the mutation stage, step (2a) from Section III is performed. The difference is that the Greedy approach picks the first best color, whereas the Genetic Algorithm picks a random "best" color. This seems like a randomized Greedy approach. by having a population size of 50, we are performing these operations on 50 graphs, each with a random starting point. With the help of randomness and best performing parents, hopefully some of the graphs converge to a solution.

A..2 Implementation Limitations

All of our code is written in python and we developed this algorithm over many iterations. The main problem with the Genetic Algorithm is the time requirements to converge upon a solution. One problem that we ran into was having to reallocate memory for every chromosome for every generation. After some profiling and modifications, we managed to optimize this problem out. The main contributing slowdown to this program is the *adjacent_colors*() method, which must iterate over every vertex and it's neighbors in order to mutate the chromosome. If this could be sped up, the Genetic Algorithm could potentially perform better. The numbers of colors presented in this report for the exam graph is usually done within a few seconds. However, in cases of trying to use a smaller number of colors, the algorithm slows down, but the fitness score does in fact continue to tend towards optimal. This leads me to suspect that a better coloring can be obtained at the cost of more time.

A..3 Parameter Tuning

The genetic algorithm presented in the paper has some interesting properties. The optimal fitness score is defined to be 0, since that would mean there are no conflicting edges. The authors present two different parent selection and mutation functions in an attempt to converge as quickly as possible. Despite trying to implement those functions, I was unable to reproduce the same benchmarks that were presented in the paper. However, I did try various ideas and with some fine tuning, I managed to get fairly decent performance. One problem that I currently have with it is the execution speed. Python isn't particularly fast, but I also am not familiar enough with the language to be able to optimize it enough. After profiling the code, I found that the bottleneck is from constantly getting the colors of adjacent edges. This also explains why the algorithm struggles with densely connected graphs.

There are some details left out of the paper as well. For example, they talk about parent selection, crossover, mutation and then adding the child to the population. However, they only add one child to the population while also mentioning that the population size should be constant. Therefore, I either need to repeat the process with another child so that I can replace both parents, or leave a parent in the population. I experimented with both. Creating two children made the fitness scores more erratic. This could potentially be beneficial since there is more exploration, but it seemed to converge onto a pattern. I think this would be a more feasible option if there was more randomness in play. The other option I explored was to take the fittest of the parents and put that parent back into the population. This had the effect of a slow, but steady decrease in the fitness score.

Bad edges are defined as edges between vertices of the same color. The fitness function is calculated

by counting the number of bad edges. However, given enough colors and a large enough graph, this is pointless. For example, if I gave the genetic algorithm enough colors for each vertex, then the algorithm would converge in seconds, usually around 12 to 20 generations. However, the color assignment seems like it might be slightly better than trying to randomly assign colors. The result of this would usually be around 500 out of 800 colors. So perhaps an adjustment that can be made is to have a penalty for chromosomes that use a lot of colors. That way the fitness score improves when there are better assignments or there are less colors used or, ideally, both.

Another thing to take into consideration is the initial population. I tried assigning colors at random while also using all of the colors, assigning at random using a subset of colors and assigning all the vertices the same color. The initial assignment with all vertices the same color converged quickly at first, but then peaks. Assigning colors at random with a color for each vertex converges in about 10-20 generations but also uses about 500 colors. Finally, assigning the vertices at random, but only using a subset of the colors behaves similar to assigning them to one color.

I also tried limiting the number of colors that were available to the algorithm. For example, giving the algorithm a set of 50 colors to work with in the exam graph of 800 vertices allows the algorithm to converge to a solution in about 20 generations. Giving it 35 colors makes it converge at around 50 to 200 generations.

Some other variables that I changed were the population size, mutation rate and crossover rate. Changing the population size didn't really help. Having a high mutation rate was good, but I found that 0.8 tends to work best with a crossover rate of 1.0. If the mutation rate was too high, it would often get to a good fitness score, but then get stuck.

Finally, I found that a mutation rate of 0.8 with a crossover rate of 1.0, population size of 50 and between 37-40 colors converges in a reasonable amount of time. However, the genetic algorithm performs poorly on random graphs since I can't really fine tune the parameters. So one of the downfalls of the genetic algorithm is that it sometimes requires fine tuning the parameters for specific examples.

X. CONCLUSION

In this paper we experimented with three graph coloring algorithms as applied to exam scheduling. Our tests have indicated that Welsh-Powell performs the best in term of speed and less number of colors. The Greedy approach performs the next best. The Genetic Algorithm, although interesting, did not perform as well for us. Solutions can be arrived at quickly if given enough colors, but it does not perform as well as Welsh-Powell or Greedy. The main take away from the Genetic Algorithm is that good performance

requires tuning the parameters. For this reason, the Genetic Algorithm performs poorly on random graphs without tuning the parameters. This is specific to each problem, so the applications of the Genetic Algorithm approach must be specific to the problem as well for best results.

REFERENCES

[1] R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, et al.. Explainable AI: the new 42?. 2nd Int. Cross-Domain Conf. for Mach. Learning and Knowl. Extraction (CD-MAKE), Aug 2018, Hamburg, Germany. pp 295-303, ff10.1007/978-3-319-99740-7_21ff. ffhal-01934928f

[2] R. R. Hoffman, G. Klein, and S. T. Mueller, "Explaining explanation for "explainable AI"" in *Sage Journals*, vol.62 no. 1, pp. 197-201, Sept. 2018. [Online]. doi:10.1177/1541931218621047

[3] R. Kass, and T. Finin, "The need for user models in generatnig exper system explanations" 1988. [Online]. https://repository.upenn.edu/cis/

[4] B. G. Buchanan, and E. H. Shortliffe, "Rule-based expert systems: The MYCIN experiments of the Stanford heuristic programming project" pp. 754 1984. [Online]. http://www.shortliffe.net/Buchanan-Shortliffe-1984/MYCIN

[5] W. Swartout, C. Paris, and J. Moore, "Explanations in knowledge systems: design for explainable expert systems". *IEEE Expert* pp. 58–64 1991. [Online]. http://people.dbmi.columbia.edu/

[6] I. Bratko, "Machine learning: between accuracy and interpretability". in *Int. Centre for Mech. Sci.* vol 382 pp. 163-167 1997. [Online]. https://link.springer.com/chapter/10.1007/978-3-7091-2668-4_10

[7] Y. Lou, R. Caruana, and J.Gehrke. "Intelligible models for classification and regression". in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery and Data Mining*, 2012, pages 150–158. ACM.

[8] M. T. Ribeiro, S. Singh, C. Guestrin, ""Why should I trust you?": Explaining the predictions of any classifier" in *IEEE Access*, vol.6, pp. 52138 - 52160, Sept. 2018. [Online]. doi: 10.1109/ACCESS.2018.2870052

[9] G. Montavon, S. Bach, A. Binder, W. Samek, K. Muller, "Explaining non-linear classification decisions with deep taylor decomposition" in *Pattern Recognition*, pp. 211–222, 2017. [Online]. https://arxiv.org/abs/1704.04133

[10] D. Kumar, A. Wong, G. W. Taylor, "Explaining the unexplained: A class-enhanced attentive response (CLEAR) approach to understanding deep neural networks" [Online]. https://arxiv.org/abs/1602.04938

[11] Z. C. Lipton, "The mythos of model interpretability" in *ICML Workshop on Human Interpretability in Mach. Learning (WHI 2016)*, 2016. [Online]. https://arxiv.org/abs/1606.03490

[12] A. Preece, "Asking 'why' in AI: explainability of intelligent systems - perspectives and challenges". [Online]. https://dais-ita.org/sites/default/files/2326_paper.pdf

[13] A. Adadi, M. Berrada, "Peeking inside the black-box: a survey on explainable artificial intelligence" in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowl. Discovery and Data Mining (KDD'16),*, pages 1135– 1144. ACM. 2016. [Online].https://dais-ita.org/sites/default/files/2326_paper.pdf

[14] D. R. Chittajallu, B. Dong, P. Tunison, R. Collins, k. Wells, et al., "XAI-CBIR: explainable AI system for content based retrieval of video frames from minimally invasive surgery videos" in *2019 IEEE 16th Int. Symposium on Biomed. Imaging (ISBI 2019)*, April. 2019. [Online]. doi: 10.1109/ISBI.2019.8759428

[15] C. A. Kulikowski, "Artificial intelligence methods and systems for medical consultation " in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-2, no. 5 pp. 464 - 476, Sept. 1980. [Online]. https://ieeexplore.ieee.org/document/6592368

[16] J. M. Long, j. R. Slagle, M. Wick, E. Irani, j. Matts, et al. "Use of expert systems in medical research data analysis: The POSCH AI project ", vol.1, pp. 796. [Online]. doi: 10.1109/AFIPS.1987.120

[17] L. Ding "Human knowledge in constructing AI systems - neural logic networks approach towards an explainable AI ", *Int. Conf. Knowl. Based and Intell. Inf. and Eng. Syst.*, Sept. 2018. [Online]. https://app.dimensions.ai/details/publication/pub.1106387404?and_facet_journal=jour.1320495

[18] D. Gunning "Explainable artificial intelligence", in *DARPA/120*, Nov. 2017, [Online]. https://www.darpa.mil/attachments/XAIProgramUpdate.pdf

[19] E. Tjoa and C. Guan "A survey on explainable artificial intelligence (XAI): towards medical XAI", July. 2019, [Online]. https://arxiv.org/abs/1907.07374

[20] F. Doshi-Velez and B. Kim "Towards a rigorous science of interpretable machine learning", Fep. 2017, [Online]. https://arxiv.org/abs/1702.08608

[21] Y. Xie, X. Chen, and G.Gao "Outlining the design space of explainable intelligent systems for medical diagnosis", March. 2019, [Online]. https://arxiv.org/pdf/1902.06019.pdf