Applications of Combinatorial Algorithms for Graph Traversal and Efficiency

Ephram Chun

May 2021

Abstract

An algorithm is a finite sequence of unambiguously defined steps that carries out a task. In this paper, we will explore the applications of combinatorial algorithms traversing through a graph. To understand the graph traversal algorithms and efficiency of them, we define and apply algorithms, the Prisoner's Dilemma, and graph theory. We use the greedy algorithm and dynamic programming to create potential solutions for assigning classes to classrooms so that students can minimize their travel distance; we then inspect the benefits and disadvantages of both approaches.

Contents

1 Introduction

In this paper, we will explore the graphs using various algorithms that will give us different ways to traverse through the graphs. Then, we apply these algorithms to derive the best way of assigning classes to classrooms in a high school so that students have to walk the least. We will map school images onto graphs that we can further analyze and compare combinatorial algorithms as they try to identify the most optimal path through the graph. Lastly, we will also consider the fairness of the solution using game theory concepts.

2 Preliminaries

In this section we will define important terms that we will need to understand for later parts of the paper. We will explore graph theory, combinatorial algorithms, and game theory to further our understanding of these concepts so that we can apply them later.

2.1 Graph Theory

We will define important terms relating to graph theory and later use graph theory to create an equivalent graph of a school map, assign classrooms to these graphs and apply it to the school's infrastructure for optimal classroom placement.

Definition 2.1. A *graph* is a diagram is one or more points, lines, line segments, curves, or areas.

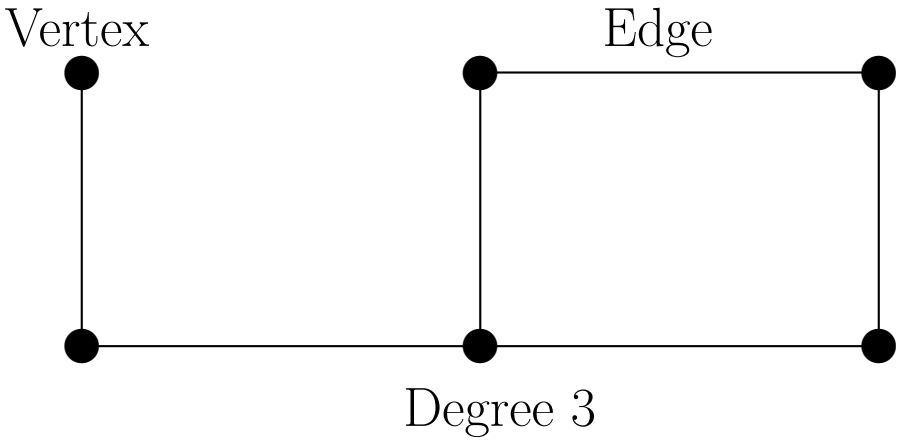
Definition 2.2. A *vertex* is a point on the graph.

Definition 2.3. An *edge* is a line connecting 2 points on the graph.

Definition 2.4. The number of edges connected to a vertex is called the *degree* of the vertex.

Definition 2.5. A *directed edge* is an edge with a direction.

Definition 2.6. A *directed graph* is a graph with vertices connected by directed edges.



2.1.1 Adjacency Matrix

Definition 2.7. An *adjacency matrix* is an $n \times n$ matrix for a graph with n labeled vertices. When $A_{i,j}$ is 1 the vertices i and j are connected by an edge, 0 if the vertices i and j are not connected by an edge. The index i represents the horizontal rows and j represents the vertical columns on the matrix.

0	A	B	C	D	E	F
A	0	0	0	1	0	0
B	0	0	1	0	1	0
C	0	1	0	0	0	1
D	1	0	0	0	1	0
E	0	1	0	1	0	1
F	0	0	1	0	1	0

This would be the adjacency matrix for the graph above which is the Lexington High School Math Building Floor 1 Graph.

Theorem 2.1. Let there be a graph with n labeled vertices and an adjacency matrix A. Then $A^k_{i,j}$ is the number of paths from i to j that are length k.

Proof. We proceed with Induction.
Base Case: $k = 1$
If $k = 1$ then $A^1_{i,j}$ would just be the adjacency matrix which by definition already shows the number of paths from i to j that are length 1.
Inductive Hypothesis:
Let us assume that the statement is true for k and prove the same for $k + 1$. Let v be any vertex of the graph. If there are $b_{i,v}$ walks from i to v, and there are $a_{v,j}$ walks of length 1. Then $b_{i,v}a_{v,j}$ would be the walks of length $k+1$ from i to j where the second to last vertice is v. Thus, the number of all walks of length $k + 1$ from i to j is $\sum b_{i,v}a_{v,j}$. We see that matrix $B = A^k$ and it follows from matrix multiplication that $BA = A^{k+1}$. □

The adjacency matrix shows numerous paths that connect 2 different vertices and this will be useful later when we need multiple paths to distribute the people's paths evenly.

2.2 Greedy Algorithm

The Greedy Algorithm is a combinatorial algorithm that has many applications. We will be comparing combinatorial algorithms while trying to optimize classrooms and the greedy algorithm is one of these algorithms.

Definition 2.8. The *Greedy Algorithm* is an algorithm that will make decisions based on the immediate reward and never returns to a previously made decision.

Example 2.1. The Greedy Algorithm can determine the minimum number of coins to make change in most currency systems. Here we have the coin currency system of the United States:

Penny	1 ¢
Nickel	5 ¢
Dime	10 ¢
Quarter	25 ¢

The Greedy Algorithm would go through the following procedure: Find the largest coin value less than the amount given and then subtract as many of that denomination we can and then repeat for the subsequent smaller denomination. However, there are scenarios where this will not work. For example, let us assume that in an imaginary society the coins represent 1-cent, 3-cents and 4-cents. Let us try to make 6 cents using the Greedy Algorithm. We would first use 1 4-cent coin and then 2 1-cent coins. However, we could have just used 2 3-cent coins which would have been less coins. One way that we can ensure that we will always find the best and most optimal solution is *dynamic programming.*

2.3 Dynamic Programming

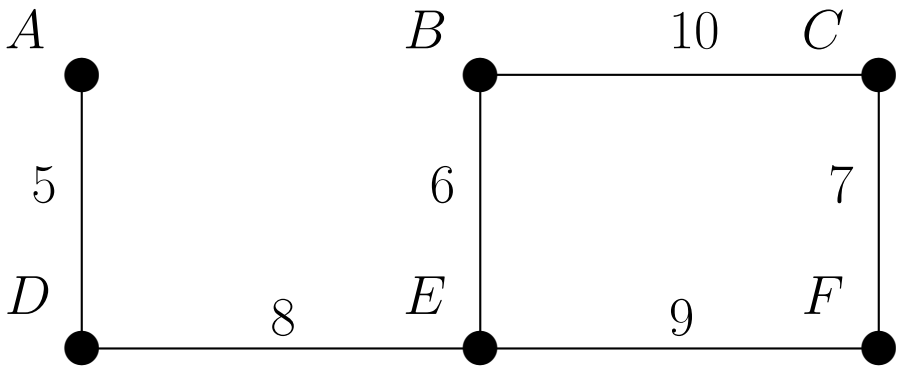
Definition 2.9. *Dynamic Programming* is an optimization over recursion, where it stores the results of subproblems and an optimization is found by a combination of these results.

Example 2.2. An example of dynamic programming is Dijkstra's algorithm which finds the shortest distance between vertices on a graph. Let us have a directed simple graph where the lengths of edges can be denoted as $d(e_i)$ which is the cost of e_i. Let us also call $\delta(v_i)$ as the length of the path with vertex v_i. as the final point. We will now split the vertex set $V(G)$ of G into 2 parts:

- Set S are the vertices which already have a path from s.
- Set T which are the vertices that do not have a direct path from s.

Thus, $S = \{s\}$ and $T = V(G) - s$. and $\delta(v) = d(s, v)$ because if there is an edge from s to v the minimum distance will be the length of that edge. Now we will find a vertex $v \in T$ for which $\delta(v)$ is minimized and we will put v into S. We will proceed with all edges leaving v and going to a vertex outside S as we proceeded with edges leaving s. Thus if vr is an edge where $r \in T$ and $\delta(v)+d(v, r) \geq \delta(r)$, then we do not do anything. Otherwise replace $\delta(r)$ by $\delta(v)+d(v, r)$ since we found a shorter path to r which consists of the shortest path to v and then vr. This is also known as we *relax* the edge vr. We iterate this process until all the vertices are in S and all the edges are relaxed. Djikstra's Algorithm finds the correct solution for any graph because it optimally explores all possibilities of paths. This is efficient because we reuse the computation from smaller paths to compute the cost for a larger path.

Example 2.3. Let us try to now use the Dijkstra's algorithm to find the shortest distance from A to C



1. Let set $S = \{A\}$
2. Relax the only edge leaving A. Set $\delta(D) = 5$. Put D in S.
3. Relax the only edge leaving D. Set $\delta(E) = 13$. Put E in S.
4. Relax the edges leaving E. Set $\delta(B) = 19$, and $\delta(F) = 22$. Put B in S.
5. Relax the only edge leaving B. Set $\delta(C) = 29$. Put F in S.
6. Relax the only edge leaving F. Set $\delta(C) = 29$. This has no effect.
7. End. Shortest path is $5 + 8 + 6 + 10 = 5 + 8 + 9 + 7 = \boxed{29}$

2.4 Prisoner's Dilemma

The *Prisoner's Dilemma* is a standard example of showing how the benefits of an individual are not parallel with society. Two criminals A and B are imprisoned in separate cells and cannot communicate with one another. The prosecutors lack the evidence to convict the pair, but they have enough to convict both on a lesser charge. Simultaneously, the prosecutors offer each prisoner a bargain. Each prisoner is given the opportunity either to betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. The possible outcomes are:

Prisoner's Dilemma Chart	**B stays silent**	**B betrays**
A stays silent	Both serve 1 year	A serves 3 years, B goes free
A betrays	A goes free, B serves 3 years	Both serve 2 years

If A and B both chose a random action then the expected years in prison would be $\frac{2+0+3+1}{4} = \frac{3}{2}$ years in prison. However, without loss of generality A decides to betray then A will spend an expected $\frac{2+0}{2} = 1$ year in prison while if A decides to remain silent will spend an expected $\frac{3+1}{2} = 2$ years in prison. Thus the better option for each person would be to betray even though the best option for both of them would to stay silent. This shows how sometimes the goals of an individual do not always line up with the goals of society.

3 Classroom Assignment Optimization

We used the Lexington High School School Maps to apply our algorithms and develop our understanding about their efficiency and performance. We made the classrooms and important turning points as vertices and the hallways and pathways between the classrooms served as the edges.

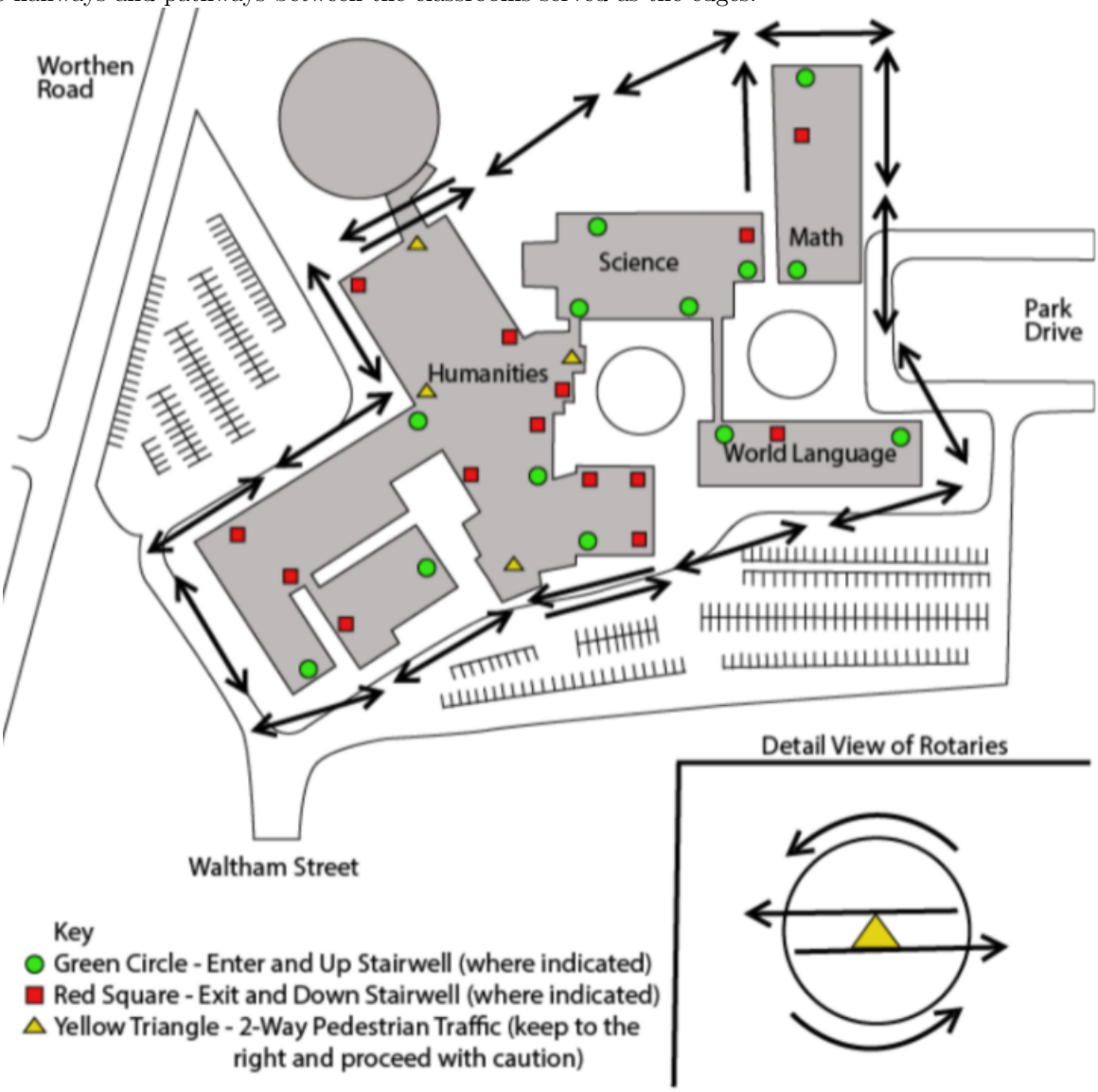


Figure 1: Lexington High School School Overhead View

3.1 Mapping onto Graphs

We will use the Lexington High School Math Building Floor 1 and we can change the classrooms into points and the hallways into edges. For our purposes, we will assume that points A, B, C, D, E, F represent only classrooms and not classes and all hallways go in both directions.

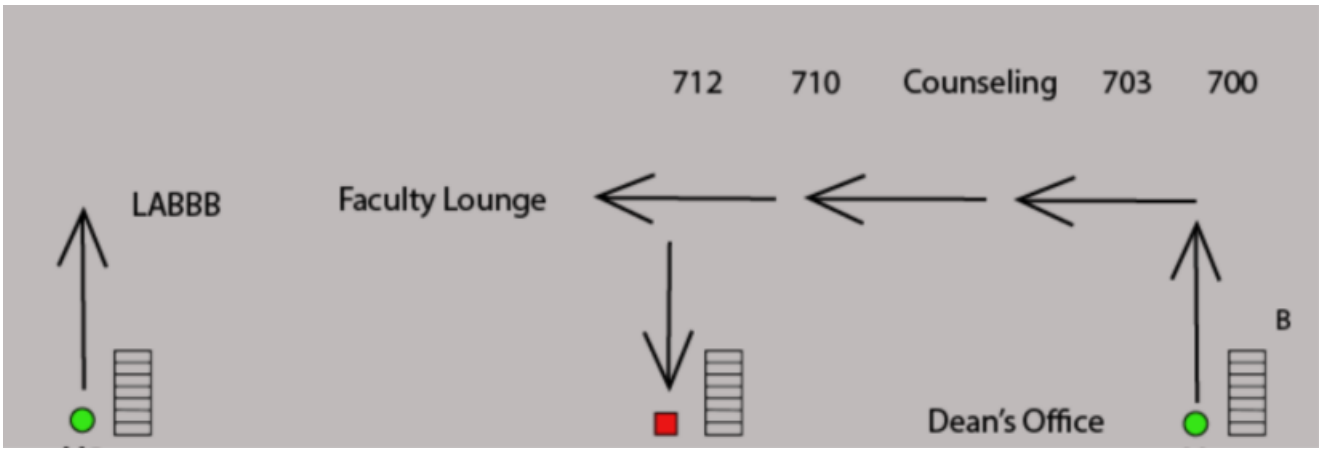


Figure 2: Lexington High School Math Building Floor 1

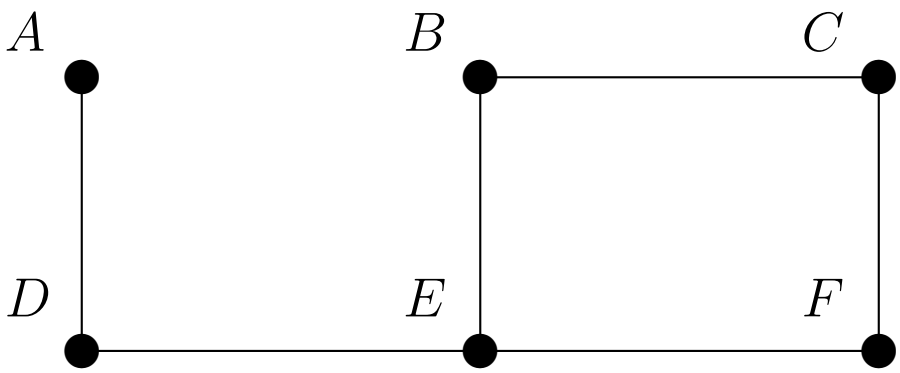


Figure 3: Lexington High School Math Building Floor 1 Graph

3.2 Assignment Problem

3.2.1 Problem Statement

We want to find the best way for a group to traverse through the school taking the most efficient route. For the sake of simplicity, we assume that we have some known number of students which is n. We further assume that each student has to attend only two classes, and they have to travel from their first class to the second class in their schedule. The total distance travelled by the students is the sum of the individual distance traveled by each student. The distance traveled by a single individual is the sum of the edges used

by the student to reach their destination. Fairness is also a concept that is involved in this problem which is balancing every student's distance such that the overall group's distance is the shortest it can be where each student gets a similar walking distance as every other student.

3.2.2 Greedy Algorithm

Let us assume that there are 6 students $S_1, S_2, S_3, S_4, S_5, S_6$ that have 2 classes each and the distances between the edges are equal.

- S_1 has classes C_1 and C_2
- S_2 has classes C_3 and C_5
- S_3 has classes C_4 and C_6
- S_4 has classes C_2 and C_4
- S_5 has classes C_6 and C_1
- S_6 has classes C_3 and C_5

We can apply the Greedy Algorithm as we iterate between each student. Specifically, we first minimize the distance for S_1 by having the classrooms be at vertices that are connected by the shortest edge in the graph. By doing so, we guaranteed the shortest possible travel distance for S_1. Then, we proceed to student S_2, by assigning their classes so that student S_2 has to walk on the minimum edge, or edges, which has not been used before.

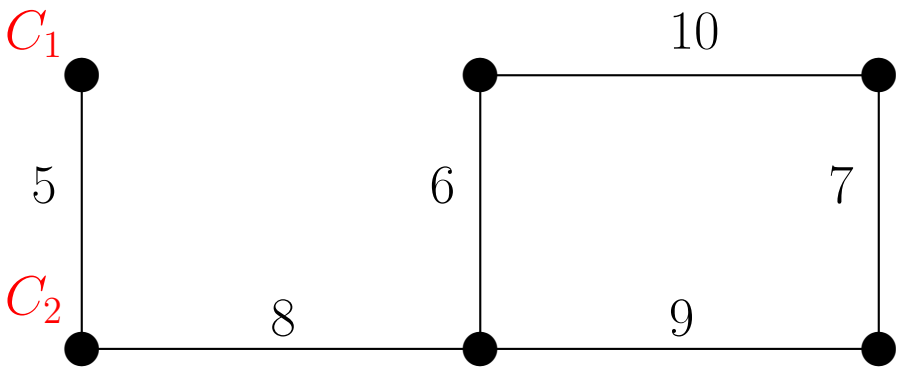


Figure 4: Iterating through Student 1

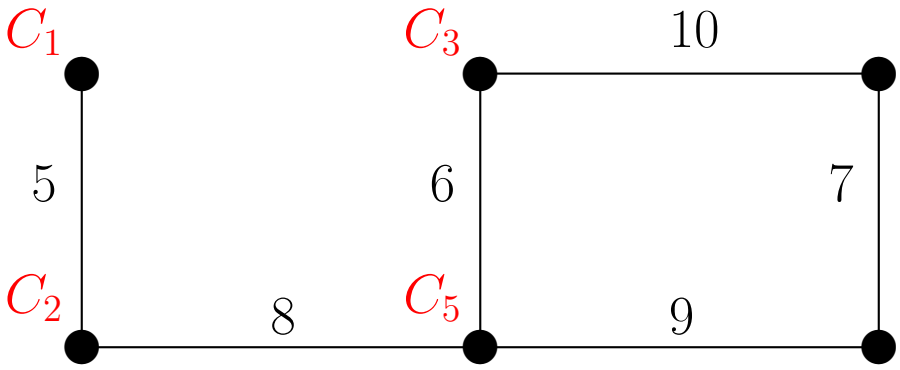


Figure 5: Iterating through Student 2

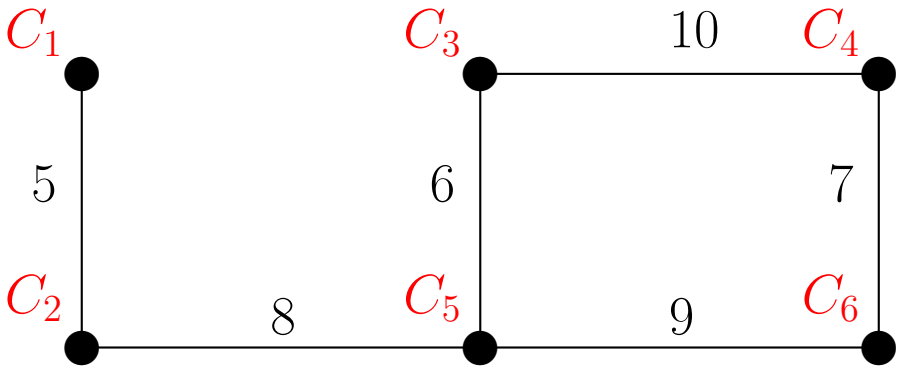


Figure 6: Iterating through Student 3

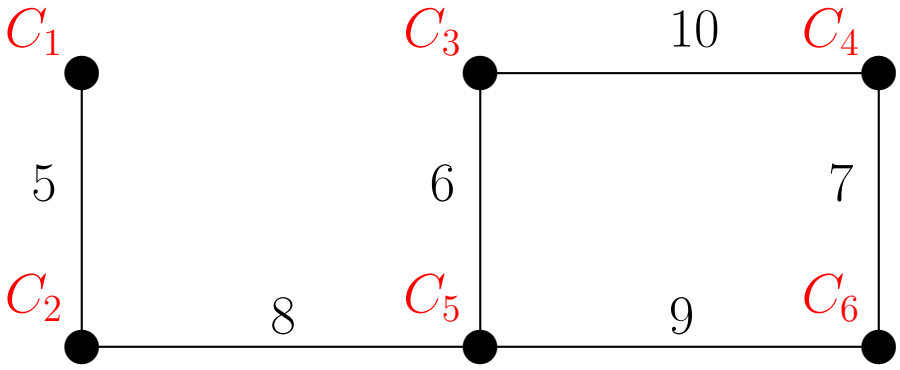


Figure 7: Iterating through Student 4,5,6

3.2.3 Dynamic Programming

Let us define α as a subset of classes and β as a subset of classrooms (vertices) and subproblem $f(\alpha, \beta) =$ outputs the best way to assign classes in α into vertices in β. Let the number of students be n and the number of classes and classrooms be m. Dynamic programming requires recursion so it is necessary to solve the problem using smaller subproblems of the same problem. To identify the best arrangement to assign m classes into m classrooms, we choose a vertex and do casework on that point where we will assign each of the

m classes into the chosen vertex. For each case, we can use subproblems from the remaining $m-1$ classes to compute the total cost for that case. Thus, we want to start this from the bottom up which means that we compute all the subproblems with $1, 2, 3 \ldots m$ classes.

Example 3.1. Let us proceed with 3 students S_1, S_2, S_3 with 2 classes each and let there be 4 classrooms.

- S_1 has classes C_1 and C_2
- S_2 has classes C_4 and C_2
- S_3 has classes C_3 and C_1

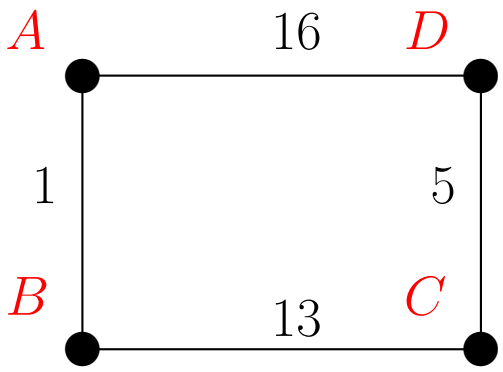


Figure 8: Sample School Format

We first choose a vertex without loss of generality let our chosen vertex be in red.

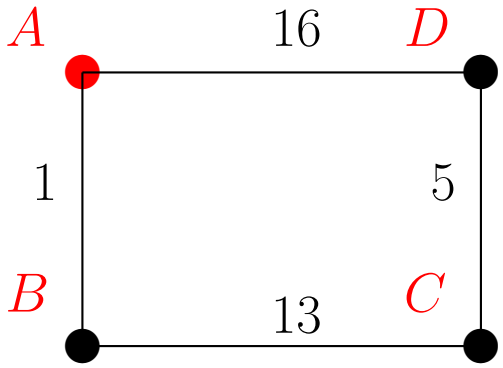


Figure 9: Sample School Format with the red point denoting our chosen vertex.

Now we can place one of C_1, C_2, C_3, C_4 into this vertex and then we will be left with the remaining 3 classrooms to place on the remaining 3 vertices. We can perform casework on which class is the best choice for the red classroom by using smaller subproblems. As defined in the previous section, subproblem $f(\alpha, \beta)$ outputs the best way to assign classes in α into vertices in β. For example, for the case where the classroom has C_1 assigned, we can compute the total distance that students have to travel by adding the output of $f(\{C_2, C_3, C_4\}, \{B, C, D\})$ and the distances students with class A in their schedule must travel. This dynamic programming solution explores all possibilities and there are 2 possible solutions so we present 1 solution in Figure 10 and the other is based on orientation.

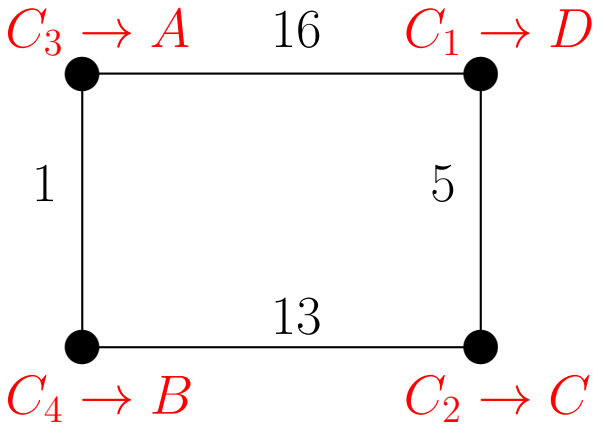


Figure 10: Optimal Classroom Assignment using Dynamic Programming

3.2.4 Evaluation of the Algorithms

The advantage of the greedy algorithm is the minimum amount of time needed to finish since it will find the best possible scenario for each student and goes through each student 1 by 1. Contrarily, dynamic programming breaks down the problem into smaller subproblems which does take significantly more time due to the numerous cases it has to execute.

One of the advantages to Dynamic programming is its accuracy because it explores every possibility and gives you the best assignments for the class placement. However, the greedy algorithm misses many possibilities because it goes through the students 1 by 1 without ever thinking back to a student or thinking ahead for other students.

Additionally, another strength of dynamic programming is its fairness. Although dynamic programming can result in an unfair assignment, it is much less susceptible as it will go through all possible cases to find the best solution. However, the Greedy algorithm will most likely not be as fair as it will give the best possible route to the first few students and those students that are assigned last would not get as short routes as the first few students.

The best case scenario for the greedy algorithm is when the computing resources are limited and we need an answer quickly. However, the unfairness in its program can become a problem in real life because some students need accommodations traveling but a solution to this is by greedily assigning the students who need accommodations first and keep going down the list following this procedure.

The best case scenario for dynamic programming occurs when there is sufficient resources and time and when we need an answer that is the best. A real life application is some classes will have a similar distribution of students from year to year and to find the best assignment for these classes we can reuse our previous traversals since their paths will be quite similar given that the classrooms are in the same place.

One consideration we must realize as we use these algorithms is crowding. The dynamic programming algorithm assigns relatively short paths for all students, but it proposes that many people traverse through a hallway, then it will get congested and slow down the entire group. This is a lesson that we can learn from the aforementioned Prisoner's dilemma where the goals of an individual do not always correspond with those of society. Although assigning an optimal path to everyone is beneficial to individuals it can also harm the group as a whole due to the slower pace it will move. To resolve this, a potential solution is to vary the costs of the paths as we assign classes to classrooms. Currently, the cost of each hallway depends on only the length but it could also vary based on its capacity. We will be able to assign classrooms for students to use the hallways fairly and efficiently.

References

[1] Bona, M. (2017) "A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory" World Scientific Publishing Co. Pte. Ltd.