Math 1060: Lecture Notes

Andrew Maurer

Summer 2017*

Abstract

Lecture notes for the section of *Math 1060: Mathematics of Decision Making* I taught in Summer 2017's through session. There were 38 class meetings, one hour each.

1 Management Science and Graph Theory

Mathematics is a tool which allows us to take complicated, noisy, real-world problems and isolate the interesting or difficult bits. In this unit, we will be concerned with the notion of a *graph*, which abstracts the notion of *binary connections*, i.e., two objects of interest are either connected or not connected. In this unit, we will see how to turn appropriate real-world problems into problems about graphs. We will also investigate some algorithms to solve efficiency problems in this context.

1.1 Urban services

Lesson 1

Goal: Identify situations where graphs will be helpful. Learn basic definitions involving graphs.
Objectives: Do examples of real world graphs. Identify connected graphs and disconnected graphs, coming from pictures and from applications.

- Talk about syllabus, course expectations etc. Office hours are a good use of time. Drop/Add ends Friday, Drop deadline is June 29.
- In this course we'll distill real-world problems into mathematical problems, and learn how to solve the mathematical problems.
- Motivating example: Meter reader starts at town hall and wants to check all meters (on foot) as efficiently as possible.
  1. Start and end at same place
  2. Walks on every side of the street with meters
  3. Repeats streets as little as possible.
- A **graph** is a mathematical object which consists of two pieces of information.
  1. Vertex information: A finite set of **vertices**.
  2. Edge information: A finite set of **edges** which connect pairs of vertices.

  In the future we may deal with **directed graphs** (**digraphs** for short) in which edges point from their start vertex to their end vertex.
- Examples of graphs and their visual representations. Note that *one graph may be drawn in many, many different ways – these are still the same graph.*
- Definitions:
  1. A **vertex** x **is connected** to a vertex y if there is a sequence of vertices, each connected to the next, which starts at x and ends with y.
  2. A **graph is connected** if every vertex is connected to every other vertex.
- Examples of real-world graphs:

*Last updated June 15, 2017.

1. Facebook / Real-World friends. Vertices are people. Edges connect those who are friends.
2. Airports and Direct flights. Vertices are airports. Edges correspond to direct flights. This is a digraph.

- Reality check:
  1. Should the airport graph be connected?
  2. If you built a friendship graph out of your graduating class from high school, would it be connected? Why or why not?
- Definitions:
  1. A **path** is a sequence of vertices, each connected to the next. We already saw this implicitly in the definition of *connected*.
  2. A **circuit** is a path that starts and ends at the same vertex.

  You'd want to find a circuit if you're planning a pilot's daily route. You'd want to find a path if you're looking for someone to introduce you to someone who knows Elon Musk.
- Back to the meter checker. Build this into a graph. Translate the problem into this:

  We are looking for a *circuit* which *passes through each edge* at least once, but no more than necessary.
- Definition: An **Euler circuit** (pronounced Oy-ler) is a circuit which passes through every edge exactly once.
- Example: the seven bridges of Königsberg. Real world problem → math problem

- Think about:
  1. Does the graph from class have an Euler circuit?
  2. Why would the graph of Königsberg not have an Euler circuit?

Lesson 2

Goals: Use Euler's theorem to identify when graphs have an Euler circuit. Use Euler's theorem to Eulerize graphs. Apply theorems correctly by thinking of them as an ingredient list.
Objectives:

- Announcement: Updated website to have yesterday's notes. This class is maleable – if there's something that interests you and you want to hear more about, let me know and I'll do my best to work it in.
- Last time: distilling problems, graphs, paths, circuits, Euler circuits, examples (abstract and real-world).
- Euler circuits in city block graph – did anyone find one? How?
- If a graph has an Euler circuit, what can we say about it?
  1. Connected – for any vertices x and y, just start at x and follow the Euler circuit until you end up at y.
  2. Every vertex has even valence – Any time the Euler circuit enters a vertex, it must also exit the same vertex.
- Back to seven bridges: why does it not have an Euler circuit?
- Theorems are mathematical statements that are always true. A theorem always has a proof, which is a logical justification for why it is true. Corollaries are theorems that are easily deduced from other theorems.

- **Theorem**: The sum of valences is twice the number of edges, i.e. $\sum$ valence = # Edges.
  Corollary: The sum of valences is always an even number.
  Corollary: There is always an even number of vertices with odd valence.

- **Theorem** (Euler's Theorem): A graph has an Euler circuit *if and only if* it is connected and each vertex has even valence.

  (The term if and only if means that this is really two statements combined into one. This is the same as saying "Any graph with an Euler circuit is connected and every vertex has even valence. Also, Any graph that is connected with every vertex having even valence must have an Euler circuit.")

  Conditional theorems – those containing the word – should be thought of as a recipe. *If I have flour, yeast, and water, then I can make bread.* If you're missing one of the ingredients, you cannot make bread.

- Practice finding Euler circuits.

- Back to the meter reader problem – if there isn't an Euler circuit, can we still solve the problem?

  Add the fewest number of *repeat* edges to "Eulerize" a connected graph. You can always do this because of the parity theorem.

- The problem of finding the smallest number of edges we must repeat to get a graph with an Euler circuit is called the *Simplified Chinese Postman Problem.*

- In the SCPP we are assuming all "streets" (a.k.a., edges) have the same length. For a regular *Chinese Postman Problem* edges can have different lengths. We may get to it in this chapter, but these will definitely come up in the next chapter.

Lecture 3

- Previously: Euler's theorem, Parity theorem. Telling whether a graph has an Euler circuit.
  Today: Finding Euler circuits and solving the Chinese Postman Problem

- **Complete graph on n vertices** (where n is a whole number). This is a graph which has n vertices, and where every vertex is connected to every other vertex. These will be important in chapter 2.

- Recall that a graph has an Euler circuit if and only if every vertex has valence an even number and the graph is connected.

- The Simplified Chinese Postman Problem asks to add the fewest number of *repeated edges* to get a graph with an Euler circuit. Phrasing this in real-world terms, "How many extra streets must our meter-reader walk down to check all the meters in the most efficient way possible?"

  The process of adding additional edges to a graph is called **Eulerizing the graph**.

- Eulerize a graph on the board.

- Euler's theorem tells us exactly where the problem lies – we must add edges to the vertices which have odd valence.

- Worksheet on Eulerizing graphs and finding Euler circuits. Collect this and use it to count participation.

- In real life, streets have lengths. We can modify our definition of graph by adding a number to each edge. Now the length of a circuit is the total length of all edges in the circuit.

- The Chinese Postman Problem is the problem of Eulerizing the graph in such a way that the resulting Euler circuit has minimal possible length.

1.2 Chapter 2: Business efficiency

Lecture 1

- Today: Hamiltonian circuits, Traveling salesman problem, algorithms, counting.

- In chapter 1 we were concerned with visiting each edge in the most efficient way possible. Now we shift to wanting to visit each vertex as efficiently as possible.
  Def. A **Hamiltonian circuit** is a circuit in a graph which visits each *vertex* exactly once.

- Why might we want to do this?

  1. Planning a vacation route – want to visit a list of cities putting minimal amounts of miles on your car.

2. Delivering several pizzas to different houses.
3. Scheduling a shuttle to take people to the airport.

- Notice that in all these applications, edges (roads) should have lengths associated to them. This leads us to investigate **weighted graphs**.
  The **cost** of a path is the sum of the weights of each edge occuring in the path.
- Rather than simply finding a Hamiltonian circuit, we may wish to find the shortest possible Hamiltonian circuit. The problem of finding the shortest Hamiltonian circuit in a weighted graph is called the **Traveling salesman problem**.
- A **tree** is a graph which has no circuits. Google "graph tree" to see what these look like. Trees appear as family trees, phylogenetic trees, and (as we'll see them) as decision trees.
- Example: Find the cost of several routes in a K_4 graph which arises from vacation planning – visiting four cities.
  Now try to just add lower cost edges instead of guessing.
  Now systematically find all circuits and their costs.
- In general there is no known "fast" way to find the lowest-cost Hamiltonian circuit. In the future we will see some ways that *usually* do a *pretty good* job at solving the TSP.
- Counting means exactly what it sounds like – finding the number of possibilities of a given situation. Sometimes this is tough.
  Fundamental Principle of counting: If you are to make n choices, with a_1 possibilities for the first choice, a_2 for the second, ... , a_n for the final choice, then there is a total of $a_1 \times a_2 \times \ldots \times a_n$ possible outcomes.
- An **algorithm** is a list of specific instructions that tells you how to solve a problem.
- Here's an algorithm to find the optimal Hamiltonian circuit:

```
List all possible circuits.
Calculate the cost of each circuit.
Find the cost of each circuit.
Choose a circuit of smallest cost.
```

Lecture 2

- Homework due Monday

  Results of questionnaire: Most said difficulty was good, most said pace was good (maybe a little fast), course is reasonably organized, and slight benefit from more examples. Seems like you'd like more worksheets and less "here's the definition of this thing." I'll do my best to have worksheets most days.

- Last time: Hamiltonian circuits, TSP, algorithms, counting.

  Today: Heuristic algorithms – greedy algorithms.

- In a K_3 there are $3 \times 2 \times 1 \times 1 = 6$ possible Hamiltonian circuits. But really these are all the same – starting position doesn't matter (triple counting) and direction doesn't matter (double counting).

  In a K_4 there are $4 \times 3 \times 2 \times 1 \times 1$ possible Hamiltonian circuits. Starting doesn't matter (quadruple counting) and direction doesn't matter (double counting) so there are really only three distinct Hamiltonian circuits.

  In a K_5 for the same reason this jumps up to 12.

  We like K_4 then, because we can find a best solution and compare the results of heuristic algorithms.

- The **nearest neighbor algorithm** is a quick algorithm to find a circuit in a complete graph that (hopefully) doesn't cost that much.

```
1. Start at a predetermined starting vertex.
2. Identify the nearest unvisited vertex.
3. Go to that nearest unvisited vertex.
4. If there are unvisited vertices remaining, return to step two.
   Otherwise, return to the starting vertex.
```

  Example in K_5.

- The **sorted edges algorithm** is a quick algorithm that might find a low-cost circuit in a complete graph.

```
1. Arrange edges of a complete graph in order of increasing cost.
2. Identify all admissible edges.
     (An edge is admissible if:
        i.  Selecting it doesn't result in three selected edges meeting at a point.
        ii. It doesn't complete a circuit that's smaller than the whole graph.)
3. Add the least-cost admissible edge to your circuit.
4. Repeat from 2. until you obtain a Hamiltonian circuit.
```

- Same example in K_5
- These are both called **heuristic algorithms** because they do not guarantee finding an optimal solution, but do appeal to our intuition. These are also **greedy algorithms** because at each stage the best choice is made, without regard for the future.
- Worksheet for the rest of class.

Lecture 3

- Last time: Heuristic algorithms to find near-solutions to TSP.

  Today: spanning trees, minimum cost spanning trees, Kruskal's algorithm.
- We say that a tree is a graph that has no circuits. (Fact: A connected tree always has $v - 1$ edges, where v is the number of vertices.)
- Motivation: Say an electrical company wants to run wires so that everyone's house is connected to "the grid" in the most efficient way possible.

  A Hamiltonian circuit may seem like a reasonable solution to this problem, but there's no reason we should want a circuit. Instead, we simply want to start at the electrical company and run (the minimum amount) of wires out to connect to everyone's home.

  Instead, we want to find a **spanning tree** for the graph, i.e., a subgraph that reaches every vertex and in which every vertex is necessary. Furthermore, we want a **minimum spanning tree**, meaning we want the total cost of this tree to be minimal.
- **Kruskal's algortithm** gives us a way to find a minimal spanning tree (there may be more than one).

```
1. Arrange edges of a graph in order of increasing cost.
2. Identify all admissible edges.
     (An edge is admissible if adding it to your tree does not result in a circuit.)
3. Add the least-cost admissible edge to your tree.
4. Stop when the tree is a spanning tree.
```

  This is *not* a heuristic algorithm because it *always* returns the minimum cost spanning tree.
- A couple examples.
- Worksheet for the rest of class.

1.3 Chapter 3: Planning and scheduling

Lecture 1

- **Oops** – meant to cover this during chapter 2. A **subgraph**, call it H, of a bigger graph, call it G, is defined by the following edge and vertex information. The vertices of H are *some* of the vertices of G. The edges of H are *some* of the vertices of G. If an edge of G is also an edge of H, both its end-points (vertices) must be vertices of H.

  A subgraph really just looks like a smaller graph inside a bigger graph.
- A little more counting: $n!$ (n **factorial**) is shorthand for $n\times(n-1)\times(n-2)\times\ldots\times2\times1$. For example $4! = 4\times3\times2\times1 = 24$. $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.

  Factorials arise when you want to count the number of ways to pick different objects until all the objects are gone. E.g., there are $n!$ Hamiltonian circuits in a complete graph on n vertices.
- Today: Scheduling tasks. List processing algorithm.

- We – some number of people, machines, or computers – want to complete some task. How do we find the best way to do it?
  1. The things doing the work are called **processors**.
  2. The problem of using these processors most efficiently is called the **machine-scheduling problem**.
- **Examples** when might scheduling be important:
  1. Scheduling in an emergency room. Need to coordinate rooms, nurses, doctors, machines, etc. in the most efficient way possible.
  2. Construction project.
- Assumptions:
  1. If a processor starts on a task, it will work until it completes the task.
  2. No processor stays voluntarily idle. i.e., if there is a processor free and a task available, then that processor immediately begins work.
  3. The requirements for ordering the tasks are given by a directed graph. (A graph with arrows instead of lines for edges.)
  4. The tasks are arranged in a **priority list** which is independent of order requirements.
- Goals:
  1. Minimize completion time of tasks that make up the job.
  2. Minimize total time that processors are idle.
  3. Find the minimum number of processors necessary to finish the job by a specified time.
- **Example**: Home construction. What do the assumptions mean? What about goals? What would be the tasks, their ordering, and the priority list?
- Example from the textbook.

Lecture 2

- Previously: Scheduling, practice scheduling.
  Today: List processing algorithm, Critical path schedules.
  Quiz on Chapter 3 on Friday. I'll have it graded in my office about a half hour after class, and solutions posted right after class.
  Test on Unit 1 (chapters 1,2,3) is on Monday. Friday is a review day.
- **List processing algorithm:** (A heuristic algorithm for assigning tasks to processors.) Call a task **ready** at a particular time if all its predecessors have been completed. At a given time, assign to the lowest-numbered free processor the first task on the priority list that is *ready* at that time and that hasn't already been assigned to a processor.
- Maybe this wasn't clear in class yesterday – the scheduling problem is to find the list of priorities that completes the job in the minimal amount of time, when following the list processing algorithm. So let's phrase it again:

  The setup of the problem is tasks (with their associated times) arranged in a digraph, with a number of processors. A proposed solution will be an ordering of the tasks into a priority list. The way we compare two proposed solutions is by applying the list processing algorithm and measuring how much time it took for the list processing algorithm to terminate on the given number of processors – this is the cost of the ordering.
- When is a schedule optimal? (Thinking about changing the *priorities*) When it completes with the smallest amount of *wasted processor time*. Equivalently, when it completes in the least amount of time.
- If we think of the priorities as fixed, we can meddle with the following variables:
  1. task times (e.g., hire better skilled workers, buy better computers)
  2. number of processors (e.g., hire more workers, buy more computers)
  3. Fewer directed edges (e.g., have someone do a task maybe they're not ready for. computer may have to do the same thing multiple times.)

- The length of a job cannot be shorter than the length of a longest path in the order requirement digraph. (Example to show this.) A longest path in the order-requirement digraph is called a **critical path**.

  Critical path scheduling is a heuristic algorithm for solving scheduling problems by assigning high priorities to tasks at the beginning of a long path. Here it is:

```
0. Start with an empty list.
1. Choose a task that heads a critical path in the order requirement digraph.
   If there is a tie, choose one of the winners in any way you wish.
2. Place the chosen task next on the list
3. Remove the chosen task from the order requirement digraph.
4. Repeat from Step 1 until there are no tasks left.
```

  The list constructed in the critical path scheduling algorithm is a priority list for the scheduling problem.

- Example & Worksheet. (Hoping I have time to make a worksheet.)

Lecture 3

- Yesterday: Critical path algorithm, list processing algorithm.
  Today: Graph coloring.
  Quiz Friday, will post solutions and email your graded quiz to you. HW1 & HW2 back tomorrow.

- Example: scheduling exams

- A **vertex coloring problem** for a graph is an assignment to each vertex of the graph a color so that any adjacent vertices (i.e., connected by an edge) are assigned different colors.
  The **vertex coloring problem** is the problem of finding a coloring on a certain graph with a certain number of colors. e.g., find a coloring of this graph with 3 colors.
  The **chromatic number** of a graph is the minimum number of colors needed to solve the vertex coloring problem.

- Two colorings:

  1. Kind of easy to find two colorings – pick any color for one single vertex. Color all adjacent vertices the opposite color, color all adjacent to those the original color, continue until either...
     (a) you get a 2-coloring of the graph.
     (b) you have two adjacent vertices that are the same color.
     In the first case, the graph has a two-coloring. In the second case, it does not.
  2. Graphs are two colorable if and only if they don't contain an odd cycle. (A cycle is a special kind of circuit where no vertex is repeated.)

- Three colorings:

  1. These are hard.
  2. Finding a three coloring is the same difficulty as the traveling salesman problem. Things of this difficulty are called NP-complete
  3. To find a three coloring you have to be inventive. It's like solving a puzzle.

- Several examples, a worksheet.

2 Voting and Social Choice

2.1 Chapter 9: Social choice: thie impossible dream

2.2 Chapter 10: The manipulability of voting systems

2.3 Chapter 11: Weighted voting systems

3 Fairness and Apportionment

3.1 Chapter 13: Fair division

3.2 Chapter 14: Apportionment

4 Further Topics

4.1 Identification Numbers

4.2 Information Science