Scheduling: The first academic paper about Penguicon

Jeffrey Quesnelle

Penguicon 2015

About Me

- Jeffrey Quesnelle, 6 year Penguicon attendee.
- Member in good standing of the U.B.S Delirium (Detroit Barfleet chapter – hope you enjoyed our parties!)
- Previous Penguicon talks: “How I Lost 100 Pounds Without Being Miserable” and “Are the Reals Really Real? A Skeptical Look at Infinity”.
- B.S. of Computer Science and B.A of Mathematics from Oakland University.
- M.S / Ph.D. Computer Science (not sure where yet though ;))
- Real life: Software engineer at automotive supplier.
- Website: http://jeffq.com
- Twitter: @jquesnelle

What’s this all about?

- In 2013 some Penguicon volunteers decided to try and create a website to streamline the process of submitting and processing events to the con.
- On this website you could submit talks, get feedback about your submission, give it tags, request resources, etc.
- Lots of cool ideas, introduced me to how Ops works.
- Got the idea, what if we set our sights bigger? Instead of just collecting the data, could we help with the actual scheduling of the con?
- It turns out this is a Hard (with a capital H) thing to do
  - This talk will explain what I mean by that!
- I spent a full semester working exclusively on this research (yikes)

Scheduling a con

- You probably know this but the logistics of pulling of an event like Penguicon can be extremely complicated!
- Dealing with the hotel, attendee registration, vendors, etc.
- Thankfully, Penguicon is blessed with a great group of contributors giving a wide range of talks, panels, and other events
  - P.S. If you play Hearthstone, come to the Fireside Gathering I'm organizing at 3 today :D
- Even after events are submitted, lots of considerations
  - What time should talks run?
  - What room should they go in?
  - Make sure presenters aren't multiply booked!
    - Multi-presenter talks get even more hairy

What we did

- Split our consideration into two problems. The first mirrored the prototypical convention scheduling problem:
  - Given presenters and a list of the talks they give (possibly with other presenters), and given the availability of rooms and presenters, find a schedule for a convention where no presenter is multiply booked
- The second was more of a what-if problem:
  - Suppose we let everyone “RSVP” for events on the site before hand. Could we also generate a schedule that minimized the number of “conflicts”?
    - A conflict exists when you RSVP for two events but they end up being held at the same time

The classic convention scheduling problem

Or, why I ended up at 12pm Sunday

Problem definition

- GIVEN:
  - A set *H* of hours
  - A set of *P* presenters, each of which are available for a subset of *H*
  - A set of *T* talks, each of which can be scheduled at a subset of *H*
  - A set of *R* rooms, which are available for a subset of *H* and are suitable for a subset of *T*
  - A list G of how many times each presenter must give each talk
- QUESTION:
  - Can we find a schedule (an assignment from of talks to rooms and hours) so that all presenters are present for all their talks, and no presenter has to be in two places at once?

A detour into computer science

- So what exactly did we study about this problem?
- One of the main branches of computer science is something called *computational complexity*
- Computational complexity isn't about studying how to solve a problem, but how "hard" this problem is to solve
- Why do we care about this?
- Computers are getting faster and faster every year
- So, we're interested in how our problem scales
  - If the problem is twice as big, does it take twice as long to finish? Four times as long? A million times as long?

An introduction to complexity classes

- “Hard” problems are those that get MUCH more difficult to solve as their size gets bigger
- Here’s some intuition:
  - Find the oldest person at the convention
    - Pretty simply, just go to everyone and ask their age, remembering who was the oldest person you’ve met up to that point. If the number of people at the convention double, you have to go to twice as many people
  - Find a group of people in the convention whose age adds up to 179
    - Even after you know everyone’s age, you have to try every different combination of people to see if their age adds up to 179. As you add more people, there are LOTS more combinations you need to try
  - Given a game of chess, determine the optimal move
    - You have to consider every possible move your opponent could take on their next turn, which involves evaluating every possible movie you could take on the turn after that, etc. etc.

Complexity classes, continued

- Computer scientists broadly group problems into “classes”. The three previous problems are each members of increasingly more difficult classes
- The first problem, finding the oldest person, illustrates the class **P**
  - Problems in **P** are those that can be solved in polynomial amount of work. For example, if there are x number of people maybe it takes x^2 amount of work
- The second problem, finding a group of people that add up to an age, illustrates class **NP**
  - Problems in **NP** are those that can be *verified* in polynomial time. There are 2^x number of possible combinations. But, if I say “these five people add up to 179”, you can quickly *verify* that the solution is correct
- The last problem, evaluating chess positions, illustrates the class **EXPTIME**
  - Problems in **EXPTIME** take an exponential amount of work, and even given a solution you can’t verify it’s correct without performing an exponential amount of work too

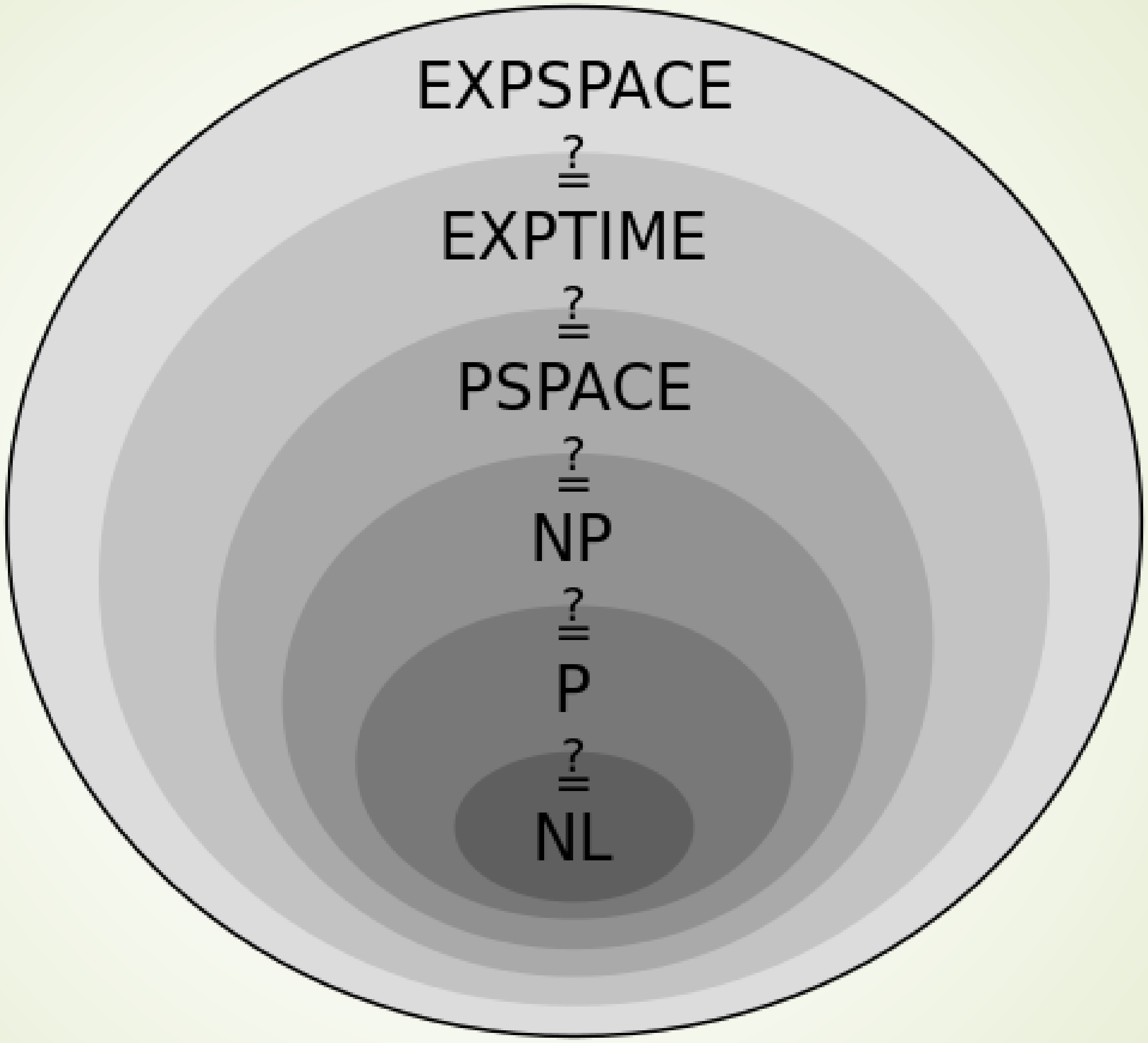
P vs NP

- The difference between solving a problem in polynomial time and exponential time is *huge*

n	n^3	2^{3n}
0	0	1
1	1	8
2	8	64
3	27	512
4	64	4096
5	125	32 768
6	216	262 144
7	343	2 097 152
8	512	16 777 216
9	729	134 217 728
10	1000	1 073 741 824

P vs NP, continued

- Why do **P** and **NP** matter? Well, **NP** has the nice property that even if solving a problem is very hard, verifying a solution is correct is easy.
- This is the basis for all cryptography that protects everything on the web (bank transactions, etc.)
  - It's easy to prove you have the correct decryption key, but very hard to find that decryption key. The most popular encryption method is known as RSA, which relies on the fact that factoring large prime numbers is hard, but proving you know the factors is easy (simply multiply them together)
- But, if we know we can verify the solution problem quickly, does that mean there must be an algorithm to solve it quickly and that we just haven't found it yet?
  - This is known as the **P** = **NP** problem, and is the most important open question in computer science
    - A proof either way currently has a bounty of $1,000,000 from the Clay Mathematics Institute


EXPSPACE
?=
EXPTIME
?=
PSPACE
?=
NP
?=
P
?=
NL

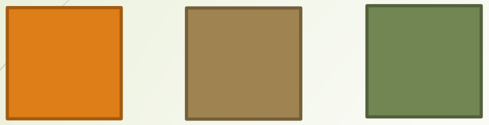
Reducibility and NP-Completeness

- When we look at a problem, it's good to know if we can even solve it at all. If the problem is firmly in **NP**, then it essentially will be impossible to solve as it gets bigger
- How can we prove that an algorithm has NO quick solution? Pretty hard to prove a negative
- This where something called **NP**-Completeness comes in
  - **NP**-Complete are the "hardest" problems in **NP**, meaning that if we can solve these problems quickly, we could solve *every* problem in **NP** quickly
- We use a technique called reducibility
  - If we can convert our problem to a problem that is already known to be **NP**-Complete, then we know that it too is at least as hard as every other **NP** problem
- There are thousands of known **NP**-Complete problems, and if we could solve ANY of them quickly, we could solve them all
  - The fact that we haven't come up with even one quick solution for any of these problems is why most computer scientists believe **P ≠ NP**

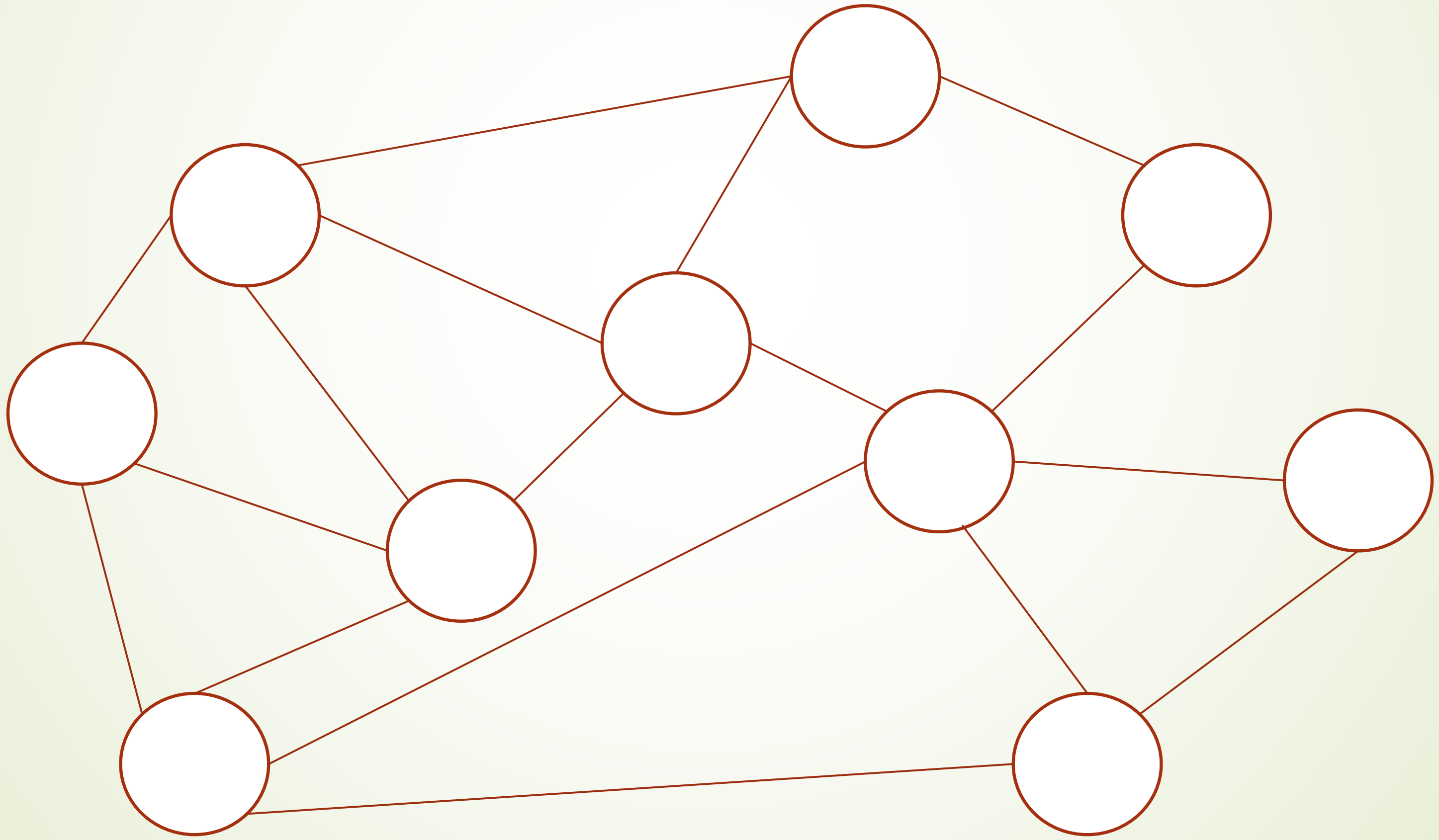
Back to our problem

- Major original result of the paper: finding a schedule for a convention is **NP**-Complete
  - This means that it's very likely no quick algorithm exists for solving it, and that the problem becomes impossible as the convention gets bigger
- How did we show this?
  - Used a reduction from graph colorability, which is known to be **NP**-Complete
    - Graph colorability asks, given a set of circles and lines between them along with a set of colors, can we color each circle so that no two circles that are connected share the same color?
    - Closely related to coloring a map of the US so no two adjacent states have the same color

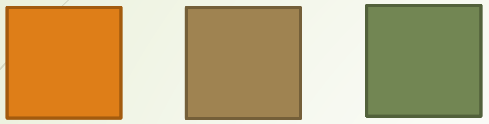
Graph colorability



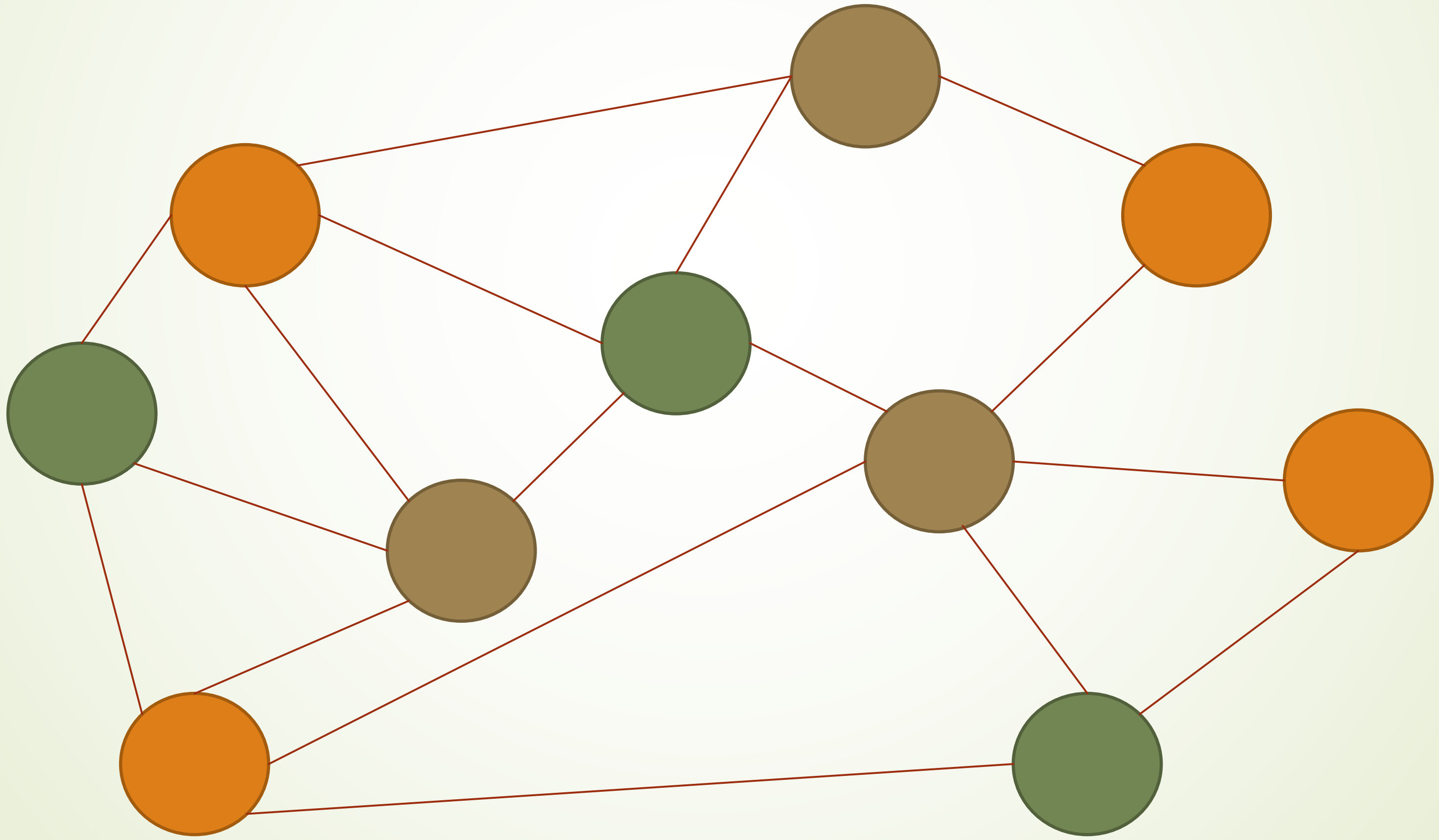
Can we color the circles with these three colors so that no connecting circles have the same color?



Graph colorability



Can we color the circles with these three colors so that no connecting circles have the same color?



Reduction from graph colorability

- For each circle, we create a “talk”
- For each color, we create an “hour” that talks can be scheduled in
- For each connection between circles, we create a “presenter”
- We now find a solution for this convention. Since no presenters (represented by connections) can be scheduled at the same hour (color), no talks (circles) that have common presenters will be assigned to same hour (color)
- So, any solution for this convention is a valid solution for the graph coloring problem, and vice versa any convention can be converted to a graph coloring problem whose solution is also a valid schedule
    - Therefore, graph colorability is reducible to our convention scheduling problem, and since graph colorability is **NP**-Complete, so is our convention problem!

What does this tell us?

- Since our scheduling problem is **NP**-Complete, it's very unlikely (unless **P** = **NP**) that there exists an algorithm that can quickly give us an optimal schedule
- There's a second result too: we don't have to come up with an algorithm to solve our particular problem! We can simply convert it to graph colorability, and then use any of the many algorithms already known to solve that, and convert the result back!
  - One of the best studied **NP**-Complete problems is known as 3-SAT, and there exists dozens of publicly available solvers that can solve 3-SAT somewhat fast for moderately sized problems
- Reducibility means everyone can focus on solving one problem, and the results can be applied to many different problems!

The RSVP problem

The RSVP problem

- We also investigated a more complex problem: what if we let everyone RSVP to events before the con? Can we create a schedule that also minimizes the number of conflicts
- Immediately we can see this is even harder! Among all possible valid schedules we have to find the one that has the least number of conflicts with (possibly thousands of) RSVPs!
- Rather than try to solve the problem ourselves, we reduced the problem to another type to be able to study it better
  - Specifically we used something called *linear programming*

Linear programming

- Linear programming involves reducing your problem to a series of linear equations.
- We can then tell a computer to either "maximize" an equation or "minimize" it, and it will attempt to find assignments of the variables that will do this
- Here's an example of reducing a problem to linear programming

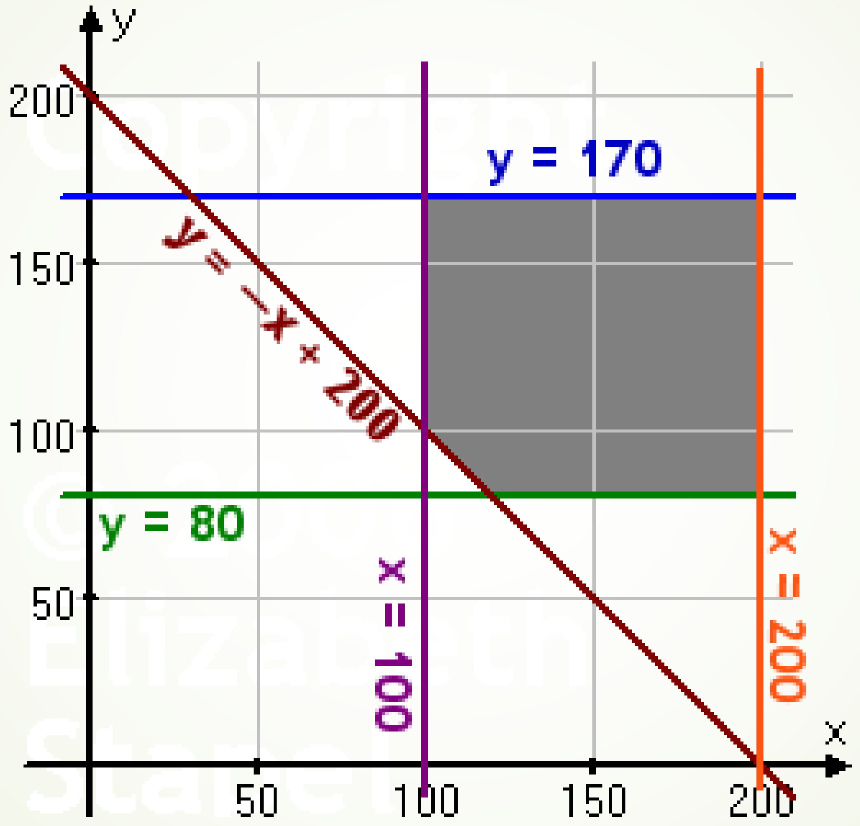
Linear programming example (from purplemath.com)

- A calculator company produces a scientific calculator and a graphing calculator. Long-term projections indicate an expected demand of at least 100 scientific and 80 graphing calculators each day. Because of limitations on production capacity, no more than 200 scientific and 170 graphing calculators can be made daily. To satisfy a shipping contract, a total of at least 200 calculators much be shipped each day.
- If each scientific calculator sold results in a $2 loss, but each graphing calculator produces a $5 profit, how many of each type should be made daily to maximize net profits?

Linear programming example, 2

- x: number of scientific calculators
- y: number of graphing calculators
- $100 \le x \le 200$
- $80 \le y \le 170$
- $x + y \ge 200$
- Maximize $-2x + 5y$

Linear programming example, 2



The gray region is the feasible region, all valid solutions are in here.
The best solution is at (100,170): -2*100 + 5*170 = 650

RSVP model

- We created an integer (linear programming with only integers) programming model that incorporated all of the constraints of a valid schedule
- We then minimized the number of the conflicts created by this schedule and let a commercial LP solver go to work
- This is the same method that airlines use to schedule planes, the NFL uses to create its schedule, etc.
- To test the model we used Penguicon 2013 data

Penguicon 2013

- 193 presenters, 253 talks, 37 hours available for scheduling, and 15 rooms

Figure 1: Presenter conflicts that must be scheduled around in PC2013

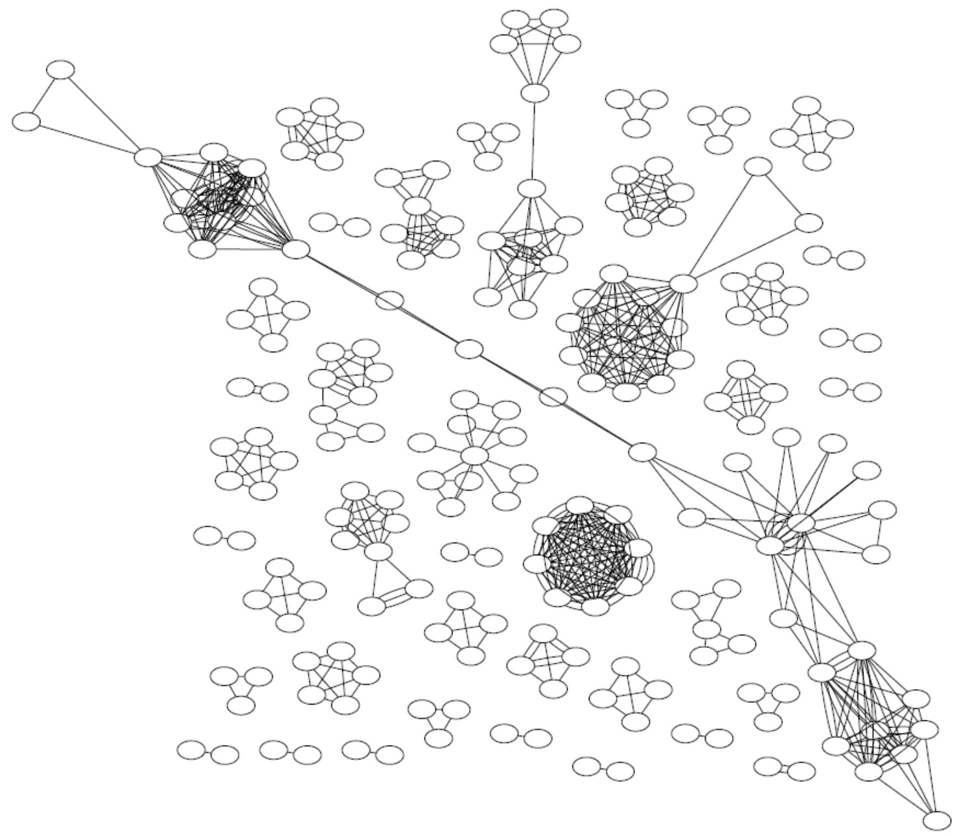
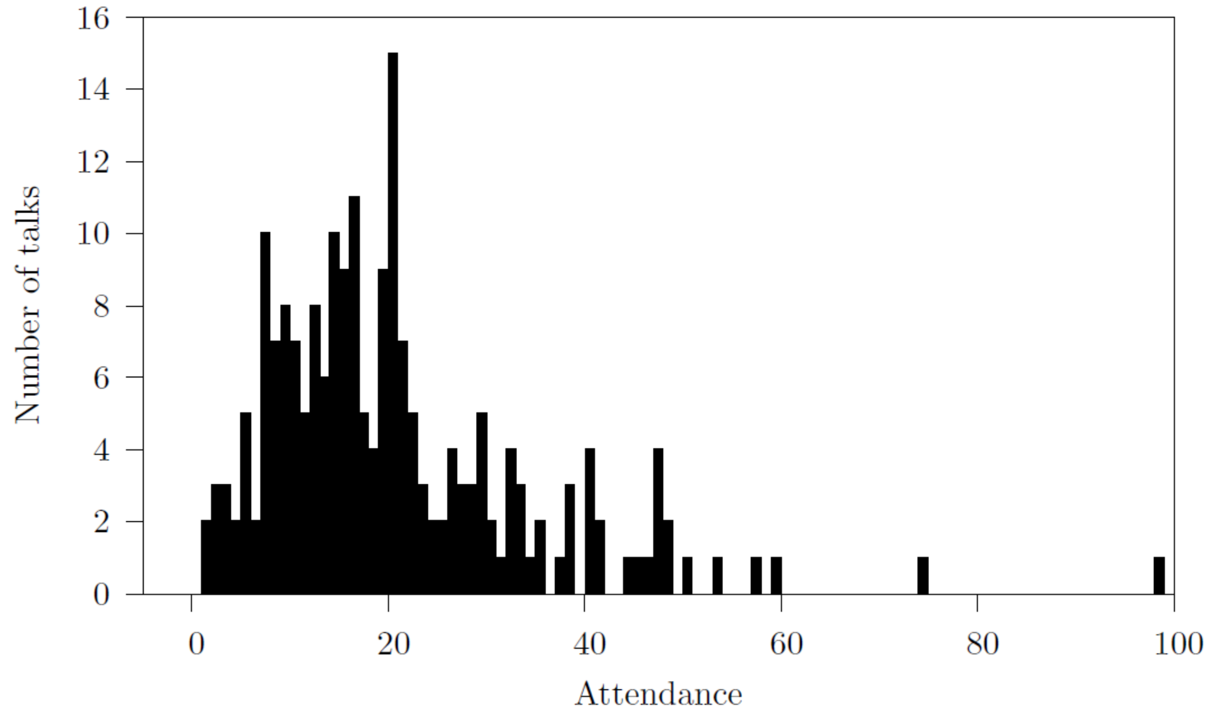


Figure 3: Distribution of attendance at PC2013



Results

- Based on attendance to talks, we randomly assigned RSVPs to 1000 (estimated con attendance) attendees. So if a talk had 30 people attend it, we randomly had 30 attendees RSVP to it
- All computations were run on a machine with 4 12-core (48 total core) Intel Xeon E5-2695 CPUs running at 2.4 GHz each with 96 GB of RAM
- Able to find a schedule with NO RSVP conflicts in 64 seconds!
  - This was using the (extremely expensive) commercial solver Gurobi, whose company was nice enough to donate a free academic license. Open source solvers were much slower

Results, 2

- What if people weren't randomly assigned? What if we changed the model so some people RSVPed to many events (the "hardcores") while lots of people only RSVPed to only a few
- Having these "hardcore" fans made the model **dramatically** harder to solve. Had to quit computation after 16+ hours
  - This follows our intuition, the more people there are with lots of RSVPs the more likely that a valid schedule will result in some of their talks being booked at the same time
  - The solver must be sure there is absolutely NO schedule with 0 conflicts before it tries to find one with 1 conflict, etc.
- We used several advanced techniques to make the model faster, including newly discovered methods known as *symmetry breaking* and *dualization*

Conclusion

- Finding schedules for conventions is hard, although for moderately sized problems (such as Penguicon) solutions are still in the realm of feasibility on high end modern hardware
- Adding the ability to RSVP to events dramatically increases the complexity of the problem
- Depending on the distribution of RSVPs amongst the attendees the problem can either be feasible, or completely hopeless, but there do exist advanced techniques to increase the speed, which leads to most realistic scenarios proving solvable
- Math is fun!

Questions?