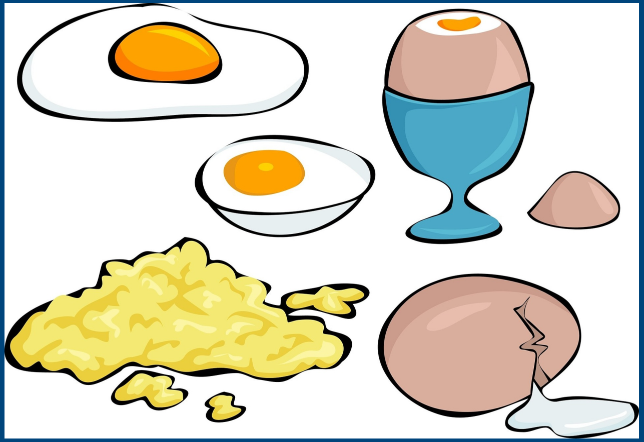
THE UNIVERSITY OF MELBOURNE

MELBOURNE SCHOOL OF ENGINEERING

Set Cover, cooked several ways

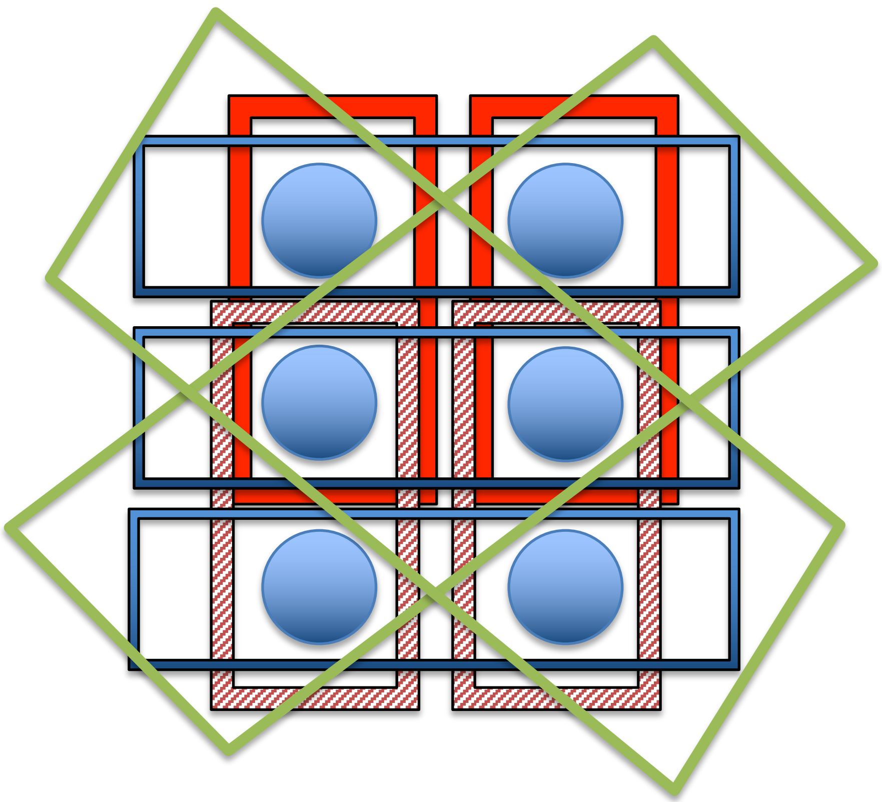
Tony Wirth

2018-10-29



What is Set Cover? Why care about it?

- Set Cover is an "old" optimization problem
- Researching classical problems is somehow satisfying
  - Part of a tradition
  - Prolific researchers considering similar questions
- Being sufficiently general, Set Cover has wide application
- With changing models of computation, we revisit some classic problems



Red: 4 Blue: 3 Green: 2

Can you see another solution with three sets?

Slightly formal definition

- Given a family $\mathcal{F}$ of sets
- Aim is to find smallest subfamily of sets that covers all items originally covered by $\mathcal{F}$
- Many variants, including weighted sets
- How much *time* and *space* required to solve Set Cover, as a function of
  - # of sets m
  - # of items n

- Let T stand for total input size (sum of set sizes)
- An example input with
  - $n = 10$
  - $m = 9$
  - $T = 28$

ABCDEF	ABCDEF
EFG	EFG
CEFIJ	CEFIJ ✓
BH	BH
CI	CI
D	D
GHJ	GHJ
ABDGH	ABDGH ✓
A	A

Applications

- Facility location
  - A hub can serve a set of demand points
  - Which hubs do we open?
- Data mining
  - Choosing a representative subset of a massive data set
  - This might be the dual problem of maximum coverage
- Retrieval
  - Topics we would like to cover
  - At least one document per topic

Bidgee [CC BY-SA 3.0]

The famous algorithm

- How might you solve this problem?
- Start with the largest set?
- Then what?

- A greedy approach
- While there are uncovered items:
  - Take the set with the largest number of (yet) uncovered items
  - “Contribution”

ABCDEF
EFG
CEFIJ
BH
CI
D
GHJ
ABDGH
B

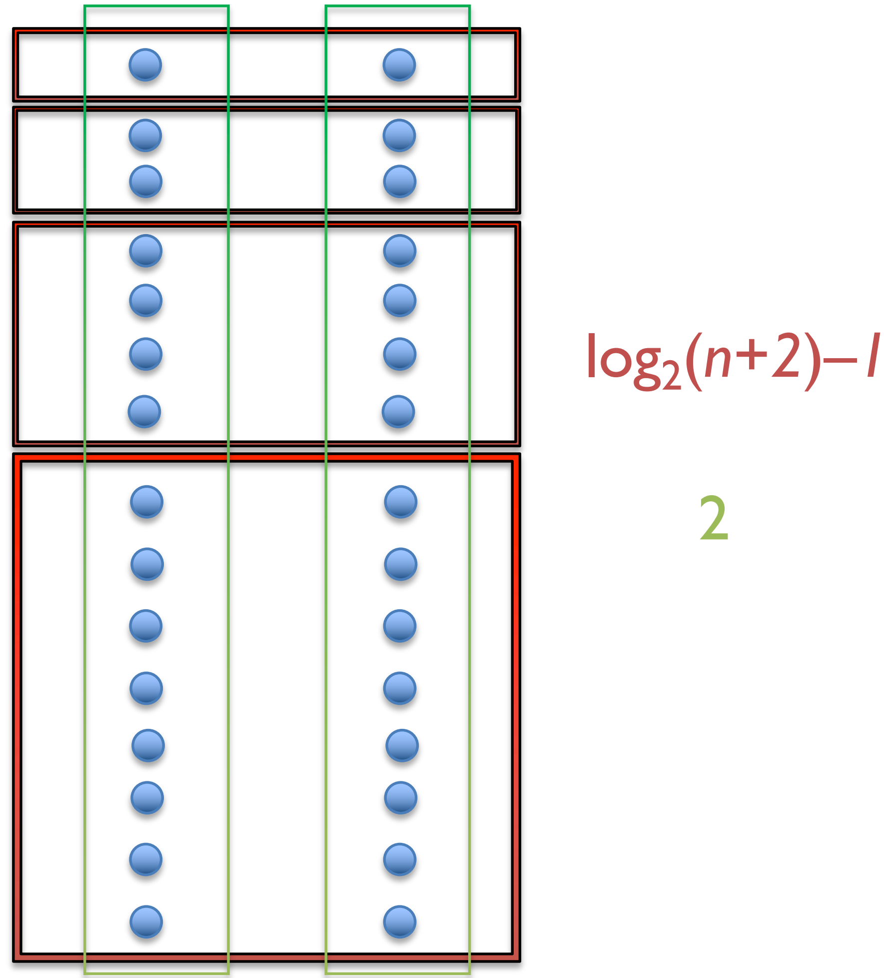
ABCDEF ✓
EF**G**
CEF**IJ**
B**H**
C**I**
D
GHJ
ABD**GH**
B

ABCDEF ✓
EFG
CEF**I**J
BH
C**I**
D
GHJ ✓
ABDGH
B

ABCDEF ✓
EFG
CEFIJ ✓
BH
CI
D
GHJ ✓
ABDGH
B

What do we know about Greedy

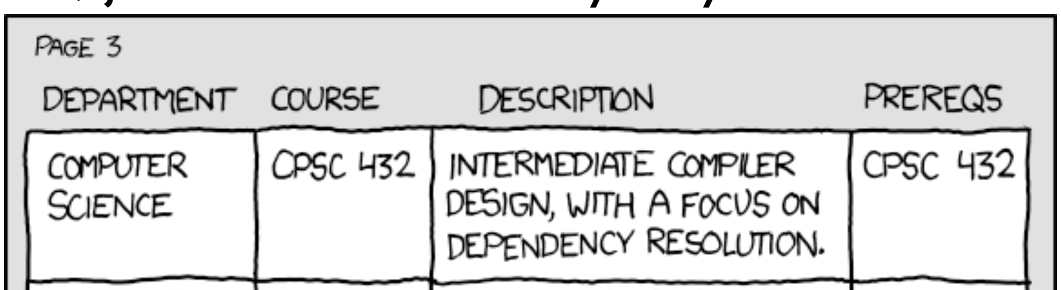
- It runs in a reasonable amount of time
- Its solution size we can prove is at most ~$\log_e n$ times optimal
  - In fact, in time polynomial in *T*, this is *best* we can do
- On most "sensible" examples, it performs within ~10% of optimal
- A "crazy" example where greedy is ~log *n* worse than optimal…



Our agenda

CC BY-SA 3.0

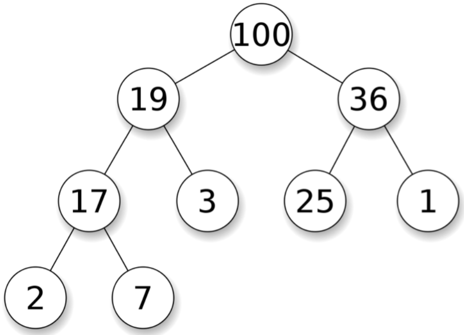
- Set Cover when data resides on disk
  - Algorithm design with AT&T Labs
  - Deeper investigation at Melbourne
- Multipass streamed instances
  - Especially lower bounds, with Dartmouth College
- Modelling software testing
  - Min Sum Set Cover with precedence constraints
  - Connections to influence maximization and community detection, joint work with U. Sydney



XKCD Dependencies
CC BY-NC 2.5

Implementing greedy on large datasets

- When reading from disk, best to read in *blocks*, not just isolated bytes
  - Reading a file sequentially especially good
  - To run efficiently, it might help to try to process sets in some sequential order
- Key issue when implementing greedy
  - When we add a set *S* to the solution
  - *Other* sets that have items in common with *S* must have their *contributions* updated to reflect their lowered importance
- Addressed with Cormode & Karloff (AT&T) and with Moffat and Lim (Melbourne)



Don't use this!

Ermishin CC BY-SA 3.0

Eager updating

- Could update counts *eagerly*:
  - Immediately know which set to add next
  - This seems to require an *index*: for each item, record its owning sets
    - As large as the input size!
  - How do we avoid too many random file accesses?

```
1:  ABCDEF       A:  18
2:  EFG          B:  1489
3:  CEFIJ        C:  135
4:  BH           D:  168
5:  CI           E:  123
6:  D            F:  123
7:  GHJ          G:  278
8:  ABDGH        H:  478
9:  B            I:  35
                 J:  37
```

Lazy updating

- Generate buckets of sets based on *initial* contribution
- [At all times, we have an *estimated* contribution of each set]
- Read each bucket in sequence, from highest estimated contribution to lowest:
  - For each set, if current contribution is same as the last estimate
  - Then add it to the solution
  - If not, *update* its contribution and append to appropriate bucket
- Assumes we store one bit per item, in fast memory, recording whether item has been covered: $O(n)$ space
- Relatively fast when sets stored on disk

Why does lazy work?

- Hopefully, it's clear that lazy updating is correct!
- Because a contribution can only *drop* as Greedy progresses, a previous evaluation of contribution remains an upper bound on current contribution
- We process sets in order of these upper-bound estimates
- If a set's actual contribution is the same as its estimate, we know it has *maximal* contribution

```
6:  ABCDEF
5:  CEFIJ ABDGH
3:  EFG GHJ
2:  BH CI
1:  D B

5:  CEFIJ ABDGH
3:  EFG GHJ
2:  BH CI
1:  D B

3:  GHJ
2:  BH CI CEFIJ ABDGH
1:  D B EFG

2:  BH CI CEFIJ ABDGH
1:  D B EFG

1:  D B EFG CI CEFIJ
```

Disk-friendly greedy

- Do we actually need a set with *largest* contribution?
- What about at least half as large as the best?
  - Proved ⇝ Set Cover solution within ~$2\log_e n$ of optimal
- Maintain sub-families of sets based on estimated contribution, *bands* of powers of 2
  - Include a set if contribution ≥ lower bound of band
  - Each sub-family has own file, accessed sequentially
  - Every second time we kick a set down, it's half the size!

```
4-7: CEFIJ ABCDEF ABDGH          4-7: ABDGH
2-3: BH CI EFG GHJ               2-3: BH CI EFG GHJ ABD
1:   D B                         1:   D B

4-7: ABCDEF ABDGH
2-3: BH CI EFG GHJ
1:   D B
```

THE UNIVERSITY OF MELBOURNE

MELBOURNE SCHOOL OF ENGINEERING

Other observations

- DFG bands need not be powers of 2
  - Trade off speed with effectiveness: e.g., factor *1.1* or *1.01*
- We considered preprocessing data, looking for items that only appear in one set (“hapax legomena”)
  - Add their sets to the solution immediately
  - Requires a few initial passes through the input
- Can prove eager approach needs only $O(T)$ time in total
- We found sequence of instances: lazy needs *at least* $T^{4/3}$ time
- In practice, lazy is much faster
- A team at Carnegie Mellon said disk-friendly greedy took more than 40 hours on one of their data sets
  - When we ran it, it took about 18 minutes!?

Ching Lih Lim's experimental results

Dataset	#Sets (m)	#Items (n)	Largest Set	Input size (T)
Social Network	37,551,359	64,961,029	3,615	1,806,067,135
UKUnion	74,117,320	126,454,248	22,429	3,376,989,142

Social Network	Eager	Lazy
Solution size	10,881,813	*10,880,876*
Time	19min52s	*2min49s*

Eager 7x slower

UKUnion	SELG	DFG (1.1)	DFG (1.01)
Solution size	*18,375,735*	18,415,017	18,381,254
Time	67min05s	*13min43s*	17min27s

LG 0.03% better, but 3.85 x slower

Multipass streams

- In the last 20 years, especially, algorithms research has focused on streaming settings
- Data arrives in a pre-determined sequence
- It should be processed immediately, and quickly
- There isn't enough space to store all the data
- Several models for this last point
  - We adopted Set Streaming: small amount of space per item
  - Necessary, just to verify a covering
- We allow **multiple** passes through the streamed data
  - Models disk access … somewhat …
  - For one pass, a factor-$O(\sqrt{n})$ approximation [Emek & Rosen]

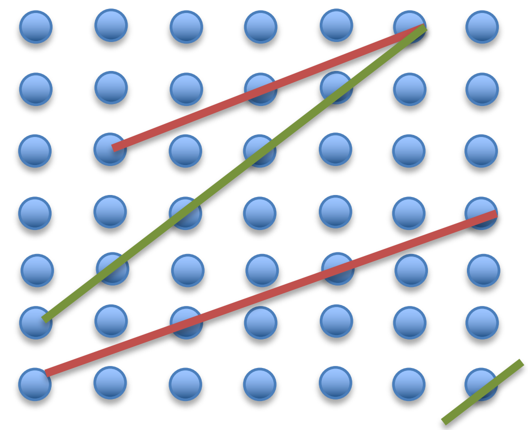
Remember this?

DFG returns!

- Given the restriction on space
  - What is *trade-off* between number of passes and approximation?
  - One pass: $O(\sqrt{n})$ approximation
  - log *n* passes: $O(\log n)$ approximation
- Say we are allowed *p* passes through the data
  - We run a generalization of DFG
  - Instead of log *n* families with lower boundaries: *1,2,4,8,16,…*
  - With *p* = 2: three families, size boundaries: $1, n^{1/3}, n^{2/3}$
  - We have *p+1* families, size boundaries, $1, n^{\frac{1}{p+1}}, n^{\frac{2}{p+1}}, \ldots, n^{\frac{p}{p+1}}$
  - Approximation factor is $(p+1)n^{1/(p+1)}$

Why is this the "right" approach?

- First: why $p+1$?
- Turns out you can run last two passes simultaneously
  - If an item isn't covered by a set of contribution at least $n^{1/(p+1)}$
  - Just record *some* set that covers it
- Rough idea of why $n^{1/(p+1)}$ is the best possible in $O(n)$ space:
- Consider a family of n **potential** sets, each a **line** $ax+b$, in $\{0,1,\ldots,q-1\}\times\{0,1,\ldots,q-1\}$, where q is prime power, $n=q^2$
  - Suppose there is a set in $\mathcal{F}$ that is the **complement** of **line** h
  - If $\mathcal{F}$ also contains **line** h, optimal solution is just 2 sets
  - If not, optimal has ≥q sets, as each pair of *lines* intersects ≤1 point



- To describe which sets are in $\mathcal{F}$ needs n bits
  - Cannot beat $\sqrt{n}$ approx
- Proof for larger p requires a different family of curves

Test case prioritization

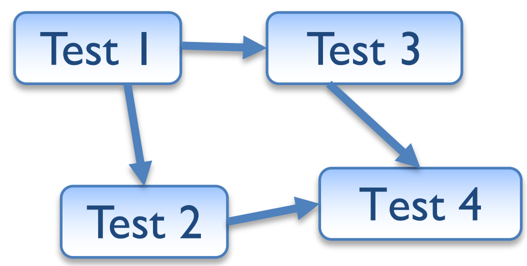
- While watching District Cricket… *cricket-Tim* discussed with me…
- We would like to test software efficiently: find faults *as soon as possible*
- We have a family of tests we could run, each covering some of the lines of code
- How do we schedule tests so that we maximize the rate of code coverage?
- What if some tests must be executed before others?

PrivateMusings - CC BY-SA 2.5

Initial progress

- We found some heuristics
  - Repeatedly choosing available test case with highest contribution
  - Sorting by contribution initially, combined with local-search to resolve dependencies
- But what exactly are we trying to optimize?
  - "Average Percentage of Faults Detected"

Test case	Fault IDs				
	A	B	C	D	E
1					✓
2		✓	✓		
3		✓		✓	
4	✓	✓		✓	



Formalizing

- Add binary precedence relation on $\mathcal{F}$, $\prec$
  - $S_2 \prec S_7$ indicates S_2 must precede S_7 (in the output)
  - We assume that the graph $\prec$ induces on $\mathcal{F}$ is acyclic
- Return **ordering** of $\mathcal{F}$ sets minimizing sum, over all elements in U, of **first** cover times

How is this like Set Cover?

- In Set Cover, aim is to find smallest sub-family of sets covering all items
- Alternative view
  - Produce **sequence** of sets
  - For each item, record earliest set that covers it (*cover time*)
  - Aim of Set Cover is to find a sequence in which largest cover time is minimized (when do we cover "last" item?)
- What if instead we minimize **sum** of cover times?

Arbitrary initial order	Greedy order
1: ABCDEF	1: ABCDEF
2: EFG	2: GHJ
3: B	3: CEFIJ
4: BH	4: BH
5: CI	5: CI
6: D	6: D
7: GHJ	7: EFG
8: ABDGH	8: ABDGH
9: CEFIJ	9: B

Cover time for each item

ABCDEFGHIJ
1111112232

Set Cover: max = 3

MSSC: total = 15
In this instance: Greedy = OPT

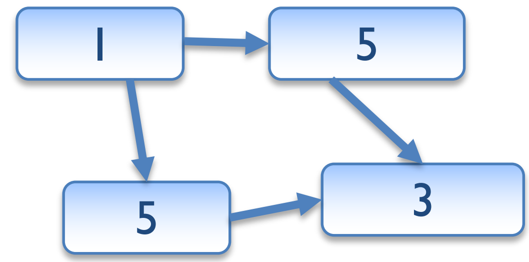
Related problems

- For *Min Sum Set Cover*, a greedy approach gives a factor-4 approximation
  - Essentially best [Feige et al.]
  - Proof uses a histogram!
- What about scheduling test cases?
- Suppose for the moment that test coverages **don't** overlap, no sense of *covering*
- Minimizing total completion time with precedence constraints has a factor-2 approximation

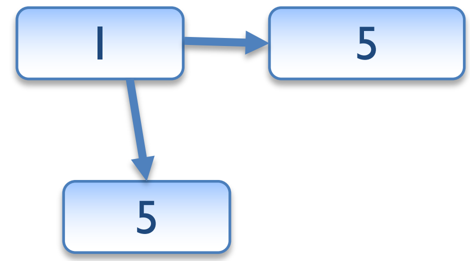
Minimizing total completion time

- Repeat the following step [CM99]:
  - Consider all subfamilies of sets that are *precedence closed*
    - $B \in \mathcal{X}, A \prec B \Rightarrow A \in \mathcal{X}$
  - Append to the schedule, and remove from input, the subfamily that maximizes ratio

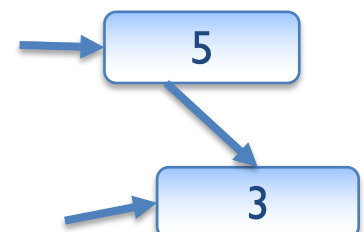
    Sum of set sizes / # sets
- Amazingly, this subproblem can be solved in polynomial time
  - Binary search over collection of max-flow/min-cut computations
  - Very similar to finding a max-density subset of graph: $|E(S)| / |S|$



Density: *3.5*



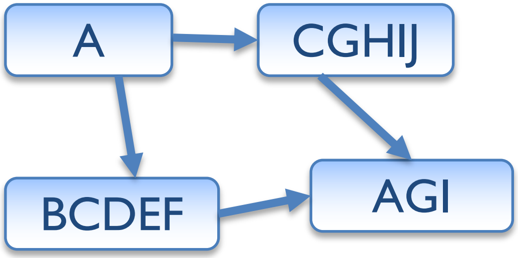
Density: $3\,{}^{2}/_{3}$



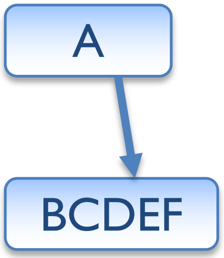
Density: *4*, but not precedence closed!

Combining coverage & completion times

- Of course, sets might in fact **overlap**
- Greedily, we choose precedence-closed subfamily maximizing density
  - Coverage of family / size of family
- This max-density precedence-closed subfamily approach "loses" factor of 4
  - Not bad, but can we solve *MDPCS*?
- Best algorithm we found was a greedy approach
  - Return set whose required sub-family has maximal density
  - Approximation factor $\sqrt{m}$, leading to overall $4\sqrt{m}$ to MSSC-Prec
  - Not very good: were we too lazy?
- On trees, approx factor = *height* + 1



Density of whole graph is 2.5

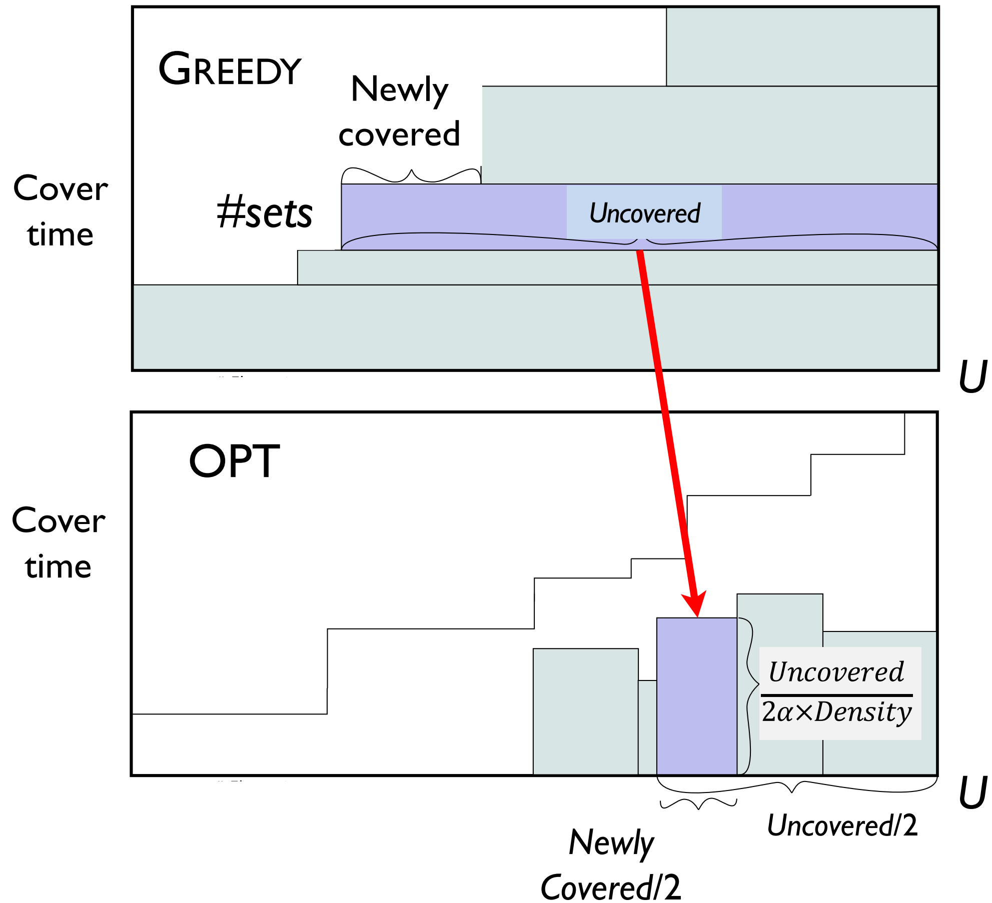


Density induced by BCDEF set is 3

OPT is 3.33

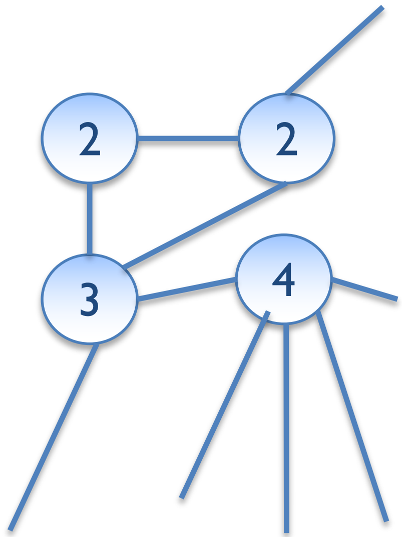
Max-Density helps solve PCMSSC

- Plot cover time (in order) for each element of *U*
  - Area under is sol'n cost
- Upper bound comprises horizontal slices of
  - height: sub-family size
  - width: #uncovered elements
- Map each slice to a column, shrunk in area by 4α (α is Max-density approx. factor)
- Since we chose sub-families greedily, up to factor α, OPT can't do *much* better
- Analysis developed from MSSC [FLT, 04]



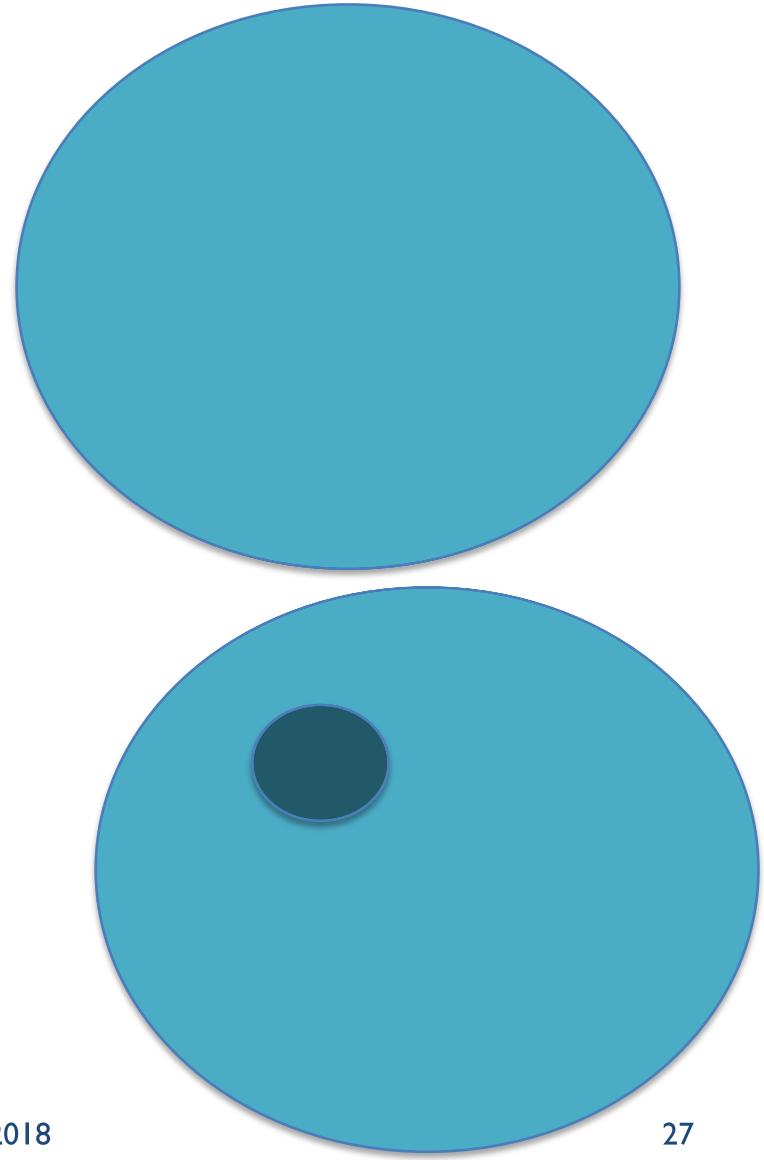
Rumor spreading

- With colleagues at Princeton & Stanford, worked on **influence maximization**, modeling rumor or infection spread
  - Each node has a resistance to infection
  - Once more neighbors than its resistance are infected, it becomes infected
- What's the smallest number of nodes we need to infect initially so that eventually everyone becomes infected?
- This is a **very** hard problem to solve well
- To prove some problem is hard:
  - We show that if we could solve it, then we could solve some other problem we know (or believe) is too hard to solve



Hardness via Planted Dense Subgraph

- In this case, source problem was Planted Dense Subgraph
- Can we distinguish between a random graph on *N* nodes in which each vertex has approximately $\sqrt{N}$ neighbors?
- And a similar random graph inserted with a random dense component on $\sqrt{N}$ nodes
  - Inside dense component, on average $\sqrt[4]{N}$ extra neighbours
- It is **strongly conjectured** that this is hard to do in a reasonable amount of time

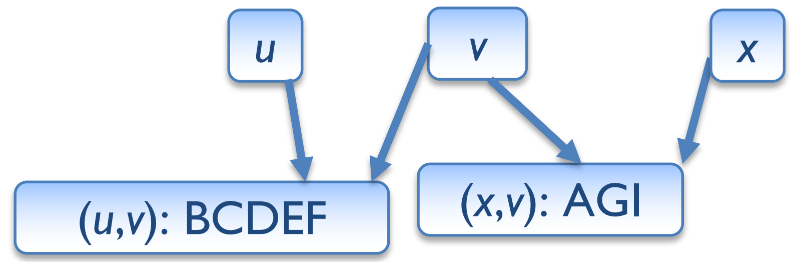


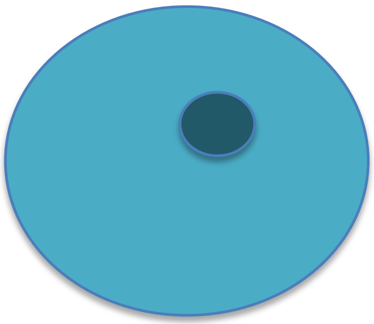
THE UNIVERSITY OF MELBOURNE

MELBOURNE SCHOOL OF ENGINEERING

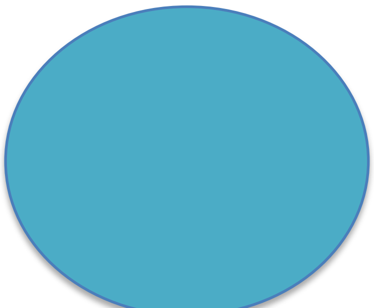
How does PDS reduce to MSSC-Prec

- Reduce a graph into an MSSC instance with precedences
- Each edge is a set covering ~ $\frac{1}{N^{3/4}}$ proportion of items
- Each vertex is a set covering "nothing", but
- Vertices *u* and *v* must **both** occur before edge (*u*,*v*)
- Set up instance so that edges in planted component *just* cover all items, hence $\sqrt{N}$ vertices suffice
  - They induce $N^{3/4}$ edges
- But if no planted component, need $N^{5/8}$ vertices to induce $N^{3/4}$ edges
- We cannot expect approximation factor for MSSC-Prec better than $n^{1/6}$ or $m^{1/12}$





My MSSC score is at most $\sqrt{N}$



My MSSC score is ~ $N^{5/8}$

Future work

- Revisit new papers from MIT, Google, Mass, and Penn
  - Partial covers
  - Set sampling and element sampling
    - Different trade-offs between space, time approximation
  - Max Coverage in streams

- What other hardness results follow from hardness of Planted dense subgraph
- With PhD candidate Xin Zhang, and Naonori Kakimura (Keio): Why **does** Greedy perform so well?

Acknowledgements

- Australian Research Council
- Colleagues
  - Graham Cormode, Howard Karloff (both then @ AT&T Labs)
  - Alistair Moffat, Ching Lih Lim
  - Amit Chakrabarti (Dartmouth)
  - Jess McClintock, Julián Mestre (Sydney)