

# A gentle introduction to community detection

Davin Choo

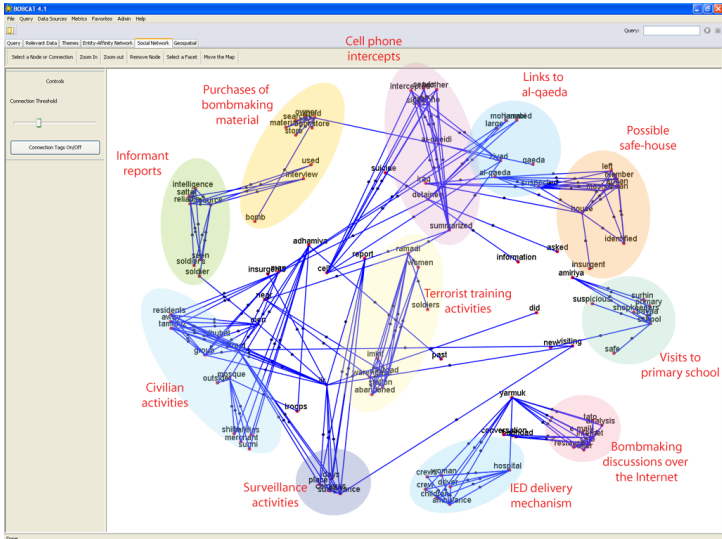
DSO National Laboratories  
CFL1

10<sup>th</sup> November 2016

# Outline

- 1 Overview
- 2 Preliminaries
- 3 Methods
  - Graph partitioning
  - Hierarchical clustering
  - Partitional clustering
  - Spectral clustering
- 4 Touch-and-go
- 5 Going further

# What is community detection?



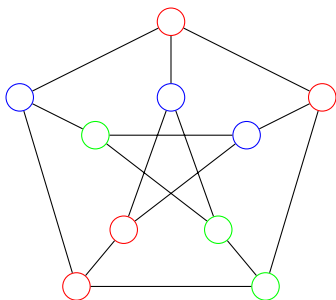
# What is community detection?

**Goal** *Quantitatively* define a community

**Hope** The *quantitative* definition captures the *qualitative* objective you have in mind

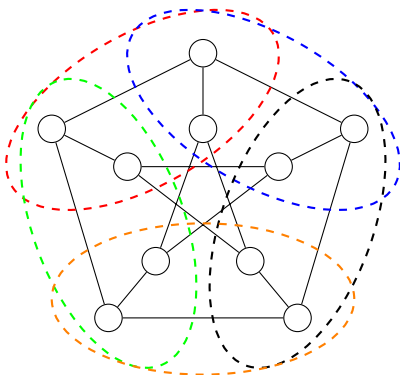
**Difficulty** Clustering is not a well-defined problem.  
Metrics are usually problem specific.  
Most clustering formulations are NP-hard

# Partitions vs. Covers



- Partition: No overlap. Each vertex only belong to 1 group
- Cover: Overlaps allowed. Can have multiple membership
- Union of either gives us all the vertices
- For this talk, will focus on partition

# Partitions vs. Covers



- Partition: No overlap. Each vertex only belong to 1 group
- Cover: Overlaps allowed. Can have multiple membership
- Union of either gives us all the vertices
- For this talk, will focus on partition

# What do we need?

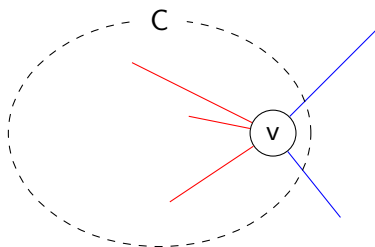
- A graph -  $n$  vertices,  $m$  edges
  - Unweighted graph: Sparse
  - Weighted graph: Weights cannot be too homogeneous
- Some concept of measure (see examples below)
  - Local measure: "Goodness" of a cluster
  - Global measure: "Goodness" of an overall partitioning
- For some algorithms,
  - Number of clusters  $k$
  - A threshold value  $d$

# A possible classification of different approaches

- Local** Form maximal groups that maintain a certain property (e.g. variants of cliques)
- Global** Maximise global partition based on a criteria (e.g. modularity)
- Vertex similarity** Group vertices based on how similar they are with respect to certain feature(s) (e.g. distance in point cloud representation)



# Degree and cluster density



- For a vertex  $v$  in cluster  $C$ ,  $deg(v) = int_v^C + ext_v^C$
- For a cluster  $C$  with  $n_c$  vertices,
  - $int^C = \sum_{v \in C} int_v^C$  and  $ext^C = \sum_{v \in C} ext_v^C$
  - Intra-cluster density  $\delta_{int}(C) = \frac{int^C}{\binom{n}{2}}$
  - Inter-cluster density  $\delta_{ext}(C) = \frac{ext^C}{n_c \cdot (n - n_c)}$
- Intuitively, a cluster *should* be a set of vertices with high intra-cluster density and low inter-cluster density

# Quality function

- Evaluate 'goodness' of a partition:  $Q(\text{Partition}) \rightarrow \text{Value}$
- Most popular: Modularity
  - $Q = \frac{1}{2m} \sum_{v_i, v_j \in V} (A_{i,j} - \frac{\text{deg}(v_i)\text{deg}(v_j)}{2m}) \mathbb{1}\{v_i \text{ and } v_j \text{ same cluster}\}$   
 $A = \text{Adjacency matrix}$
  - Compare partitioning in actual graph against a null model (randomly distribute edges).
  - Higher modularity value  $\Rightarrow$  Better community structure (?)

# The plan for today

- Graph partitioning (Kernighan-Lin, Spectral bisection)
- Hierarchical clustering (Agglomerative, Divisive)
- Partitional clustering (k-means)
- Spectral clustering

Due to time constraint,

- Details and examples only for some methods
- We can discuss in-depth after the talk :)



# Graph partitioning

**Goal** Cut up the graph into 2 parts

**Pros** Can be efficient and fast

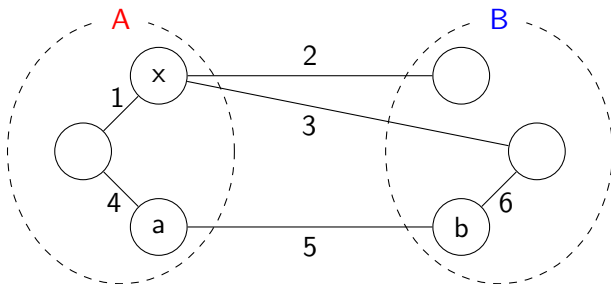
**Cons** Not natural to always cut into 2.

Need to know number of clusters  $k$

Some methods can be extended to allow multiple cluster cuts, but those methods have poorer run time.



# Kernighan-Lin (1970)



- For now, consider only single element swaps<sup>1</sup>
- Gain of moving element  $x$  (e.g. from  $A$  to  $B$ ):  

$$D(x) = \sum_{(u,x) \in A} w(u,x) - \sum_{(v,x) \in B} w(v,x) = 2 + 3 - 1 = 4$$
- Gain of swapping 2 items (e.g.  $a \in A, b \in B$ ):  $D(a,b) =$   

$$D(a) + D(b) - 2 * w(a,b) = (5 - 4) + (5 - 6) - 2 * (5) = -10$$

<sup>1</sup>In general, works with any subset size. Larger subsets  $\Rightarrow$  slower run time

# Kernighan-Lin (1970)

---

## Algorithm 1 Kernighan-Lin( $G = (V, E)$ )

---

```

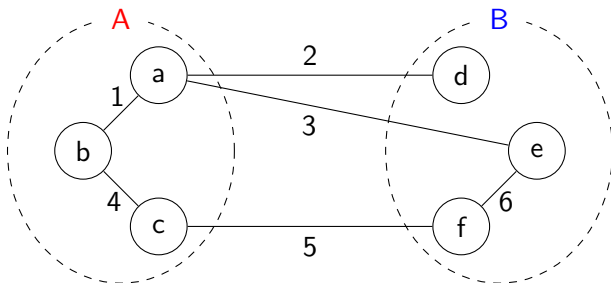
1: Initialise partitions  $A$  and  $B$ 
2: loop                                ▷ Store copy of originals  $A, B$  somewhere
3:   for  $i = 1, \dots, n$  do           ▷ Modify working copies of  $A$  and  $B$ 
4:     Compute  $D(x)$  for all  $x \in V$ 
5:      $a, b \leftarrow \operatorname{argmax}_{a \in A, b \in B} \{D(a, b)\}$            ▷ Swap  $a$  and  $b$ 
6:      $S[i] \leftarrow (a, b), g[i] \leftarrow D(a, b)$            ▷ Record for later
7:   end for
8:   if  $\operatorname{argmax}_k \sum_{i=1}^k g[i] > 0$  then           ▷ Best prefix changes
9:     Permanently apply changes  $S[1], \dots, S[k]$  to originals
10:  else
11:    return  $A, B$ 
12:  end if
13: end loop

```

---

## Kernighan-Lin (1970) tracing: 1/4

$$\text{Cut size} = 2 + 3 + 5 = 10$$



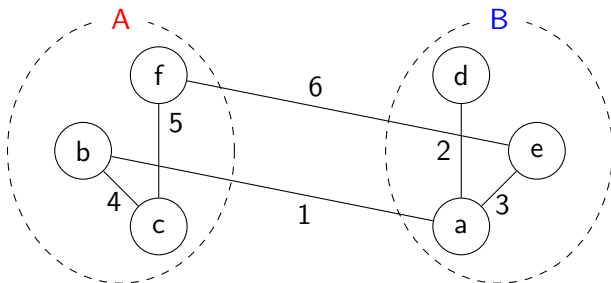
$$\begin{aligned} D(a) &= 4 \\ D(b) &= -5 \\ D(c) &= 1 \end{aligned}$$

$$\begin{aligned} D(d) &= 2 \\ D(e) &= -3 \\ D(f) &= -1 \end{aligned}$$

$D(a, f) = 3$  is the largest  $\rightarrow$  Swap  $a$  and  $f$

## Kernighan-Lin (1970) tracing: 2/4

$$\text{Cut size} = 6 + 1 = 7$$



$$D(f) = -2$$

$$D(b) = -3$$

$$D(c) = -9$$

$$D(d) = -2$$

$$D(e) = 3$$

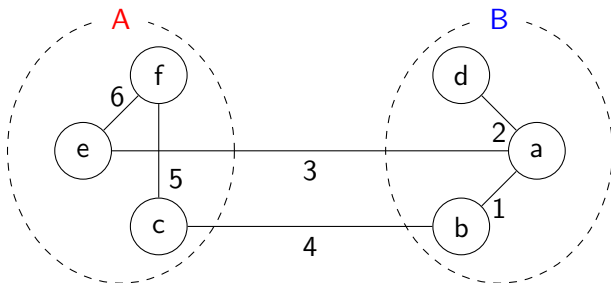
$$D(a) = -4$$

$D(b, e) = 0$  is the largest  $\rightarrow$  Swap *b* and *e*



## Kernighan-Lin (1970) tracing: 3/4

$$\text{Cut size} = 3 + 4 = 7$$



$$D(f) = -11$$

$$D(e) = -3$$

$$D(c) = -1$$

$$D(d) = -2$$

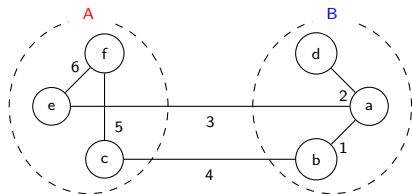
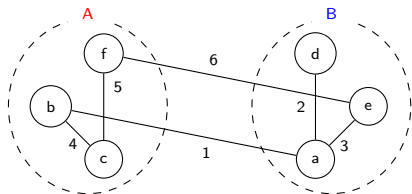
$$D(a) = 0$$

$$D(b) = -3$$

$D(b, e) = -1$  is the largest  $\rightarrow$  Swap  $a$  and  $c$

# Kernighan-Lin (1970) tracing: 4/4

- Since  $D(a, f) = 3$ ,  $D(b, e) = 0$ , and  $D(b, e) = -1$ , the best prefix sum gives us either the 2<sup>nd</sup> or 3<sup>rd</sup> graph
- Both yield cut size of 7



# Spectral bisection

- (Unnormalized) Laplacian matrix  $L = D - A$   
 $D =$  diagonal degree matrix  
 $A =$  adjacency matrix
- If  $G$  is connected, smallest eigenvector  $\lambda_1$  of  $L$  is 0
- Fiedler vector (1973):  
Eigenvector  $V_2$  corresponding to  $2^{nd}$  smallest eigenvalue  $\lambda_2$
- Bi-partition using Fiedler vector by
  - Sign of values in  $V_2$  (positive vs. negative)
  - Average of values in  $V_2$  (above vs. below average)

# Demo

See IPython notebook

# Hierarchical clustering

**Goal** Given a vertex/cluster similarity metric, iteratively join or split up vertices

**Pros** Do not assume  $k$

**Cons** Hierarchy may not be natural.  
Similarity computation may be expensive

Dendrogram is a useful way of visualising outputs of hierarchical clustering methods.

# For a suitable metric $f$

Agglomerative (Bottom-up):

- 1 Initialise every vertex as own cluster
- 2 Compute  $f(i, j)$  for clusters  $i$  and  $j$  (may set  $-\infty$  if no edge)
- 3 Combine clusters  $\operatorname{argmax}_{(i, j)} f(i, j)$  with highest  $f$  score.
- 4 Repeat previous 2 steps until only 1 cluster remain

Divisive (Top-down):

- 1 Compute  $f(\cdot)$  for all edges
- 2 Remove  $\operatorname{argmax}_e f(e)$ . Handle ties randomly
- 3 Repeat previous steps until no more edges

Quality of split depends on  $f$ , but  $f$  cannot be too expensive!

# Agglomerative (Bottom-up)

Ways to combine clusters  $C_1$  and  $C_2$ :

- Single-linkage (min):

$$f(C_1, C_2) = \min_{i \in C_1, j \in C_2} f(i, j)$$

- Complete-linkage (max):

$$f(C_1, C_2) = \max_{i \in C_1, j \in C_2} f(i, j)$$

- Average-linkage (avg):

$$f(C_1, C_2) = \frac{1}{|C_1| \cdot |C_2|} \sum_{i \in C_1, j \in C_2} f(i, j)$$

# Divisive (Top-down)

One popular algorithm: Girvan and Newman (2002)

- Lots of modifications and extensions
- Their metric  $f$  is the concept of *betweenness*  
Roughly: How frequent an edge is involved in “some process”

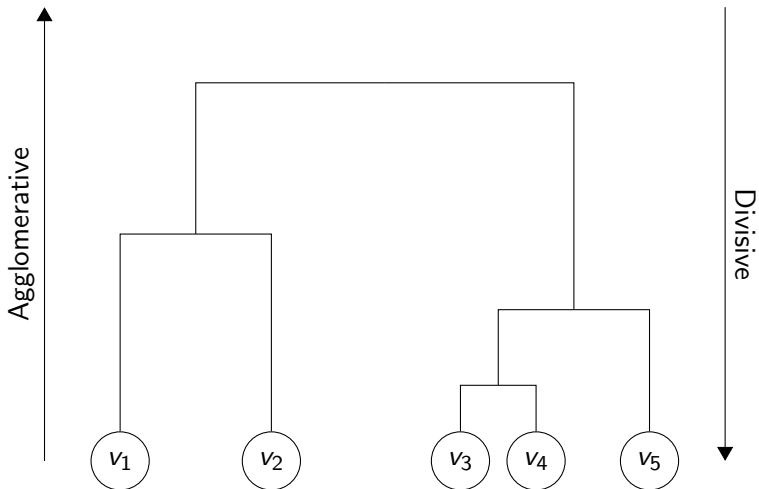
3 variants of edge betweenness

- 1  $f_1(e)$ : Geodesic edge betweenness  
# shortest paths between all vertex pairs that include edge  $e$
- 2  $f_2(e)$ : Random-walk edge betweenness  
How likely is  $e$  involved in a random walk from  $s$  to  $t$ ?
- 3  $f_3(e)$ : Current-flow edge betweenness  
Put voltage across 2 vertices  $\rightarrow$  Kirchoff's equations.  
 $f_3(e) =$  Average current of  $e$  across all vertex pairs.

Equations of  $f_2$  and  $f_3$  shown to be equivalent.



# Dendrogram example



# Partitional clustering

- Goal** Given a distance metric, separate vertices into clusters based on some cost function involving distances between points in a cluster, or points to a cluster centroid
- Pros** Fast convergence
- Cons** Need to know  $k$ . Sensitive to initialisation

# k-means

Iteratively improve from random initialisation of  $k$  centroids

- 1 Assign vertex to closest centroid
  - 2 Update centroid to average coordinate of all assigned vertices
  - 3 Repeat previous steps until convergence
- A special case of Expectation-Maximisation (EM) algorithms
  - Multiple ways to define convergence (can be a mixture):
    - Fixed number of iterations
    - Assignments to clusters did not change
    - Clusters did not change positions
    - Decrease in the sum of distances from vertices to assigned centroids is below a threshold

# Demo

See IPython notebook

# Spectral clustering

- Goal** Using a similarity metric, partition sets into clusters using eigenvectors of matrices
- Pros** Induced metric space tends to reveal clustering properties better
- Cons** Computation of eigenvalues and eigenvectors may be expensive for large graphs
- Strongly related to perturbation theory, graph cuts, etc.
  - Can view as a non-linear graph transformation / dimension-reduction preprocessing step before executing standard techniques like k-means
  - Unnormalised Laplacian matrix  $L = D - A$  (Fiedler)
  - Symmetric normalised Laplacian matrices  $L = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$  (Andrew Ng, Michael Jordan, Yair Weiss, etc.)

# Spectral clustering with unnormalised Laplacian

- 1 Compute eigenvectors and eigenvalues of Laplacian  $L = D - A$
- 2 Pick  $k = \operatorname{argmax}_{i=2,3,\dots,n} |\lambda_i - \lambda_{i-1}|$
- 3 Graph transformation:  
Form new matrix  $M = (V_1, V_2, \dots, V_k) \in \mathbb{R}^{n \times k}$
- 4 Run k-means on  $M$ , treating each row as a point
- 5 Cluster original points according to k-means results on  $M$

# Demo

See IPython notebook

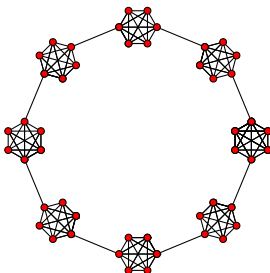
# Modularity

- Popularised by Newman and Girvan
- Recall modularity:  $Q = \frac{1}{2m} \sum_{v_i, v_j \in V} (A_{i,j} - \frac{\text{deg}(v_i)\text{deg}(v_j)}{2m}) \mathbb{1}_{\{v_i \text{ and } v_j \text{ same cluster}\}}$
- Assumption: Higher  $Q \Rightarrow$  Better partition
- Optimize  $Q$  to find best partition via methods like greedy agglomeration, simulated annealing, etc.
- Caveat:
  - The assumption doesn't always hold
  - *"Modularity maximum of a graph reveals a significant community structure only if it is appreciably larger than the modularity maximum of random graphs of the same size and expected degree sequence."*
  - See survey paper, Section IV. C. 'Limits of modularity'



# Dynamic methods

- Spin models
  - Popular model in statistical mechanics: Potts model
  - Each vertex can hold a different state/spin
  - Goal: Minimise energy  $\mathcal{H}$  based on neighbour interactions
- Random walk
  - Community structures have high density of internal edges
  - Random walkers spend long time within the same community



Note: Not dynamic in the sense of a changing graph

# Statistical inference methods

- Find best hypothesis that fits actual graph topology
- Bayes theorem:  $P(\Theta|D) = \frac{1}{Z}P(D|\Theta)P(\Theta)$ 
  - $\Theta$ : Parameters/hypothesis
  - $D$ : Data/Actual graph
  - $Z$ : Normalizing constant (usually hard to compute)
- Methods usually find  $\Theta$  that maximise  $P(D|\Theta)$ .
- Example: Planted partition model (Hastings, 2006)

# Planted partition model (Hastings, 2006)

**Input** Graph  $G$ , number of clusters  $k$ ,  $p_{in}$  and  $p_{out}$

**Solve**  $\operatorname{argmax}_{\text{partition } q_i} P(G|q_i)$  via belief propagation

**Output** Most likely partition  $q^*$

- $p_{in}$ : Probability that vertices in same group are linked
- $p_{out}$ : Probability that vertices in different groups are linked
- Notice similarity to spin model approach.

# What's next?

Today:

- Graph partitioning:  
Kernighan-Lin, Spectral bisection via Fiedler vector
- Hierarchical clustering  
Agglomerative, Divisive (Girvan and Newman)
- Partitional clustering: k-means
- Spectral clustering with unnormalised Laplacian

Other interesting directions:

- Finding covers (e.g. clique percolation)
- Multiresolution and cluster hierarchy
- Detection of dynamic communities

# How to develop a clustering algorithm?

No single best algorithm for all problem settings.

- Make good observations in your problem domain:  
Construct and study a few graphs, extract insights, etc.
- Formalise the observation quantitatively
- Find ways to optimise
- Test and check that your *quantitative* measure correctly reflects your *qualitative* goal

## Further reading

- Community detection in graphs

<https://arxiv.org/pdf/0906.0612v2.pdf>

- A Tutorial on Spectral Clustering

<https://arxiv.org/pdf/0711.0189v1.pdf>

- On Spectral Clustering: Analysis and an Algorithm

<http://ai.stanford.edu/~ang/papers/nips01-spectral.pdf>