Clustering

Outline

Topics and Learning Objectives

- Discuss clustering applications
- Cover the greedy, Max-Spacing K-Clustering Algorithm

Exercise

- Clustering practice

Extra Resources

- Algorithms Illuminated Part 3, Chapter 15

Clustering

Goal: given a set of n "points" we should group the points in some sensible manner

What are some possible sets of points?

- Webpages, images, genome fragments, people, etc.

For anyone interested in machine learning, clustering is a *relative* of unsupervised learning

Clustering

Assumptions:

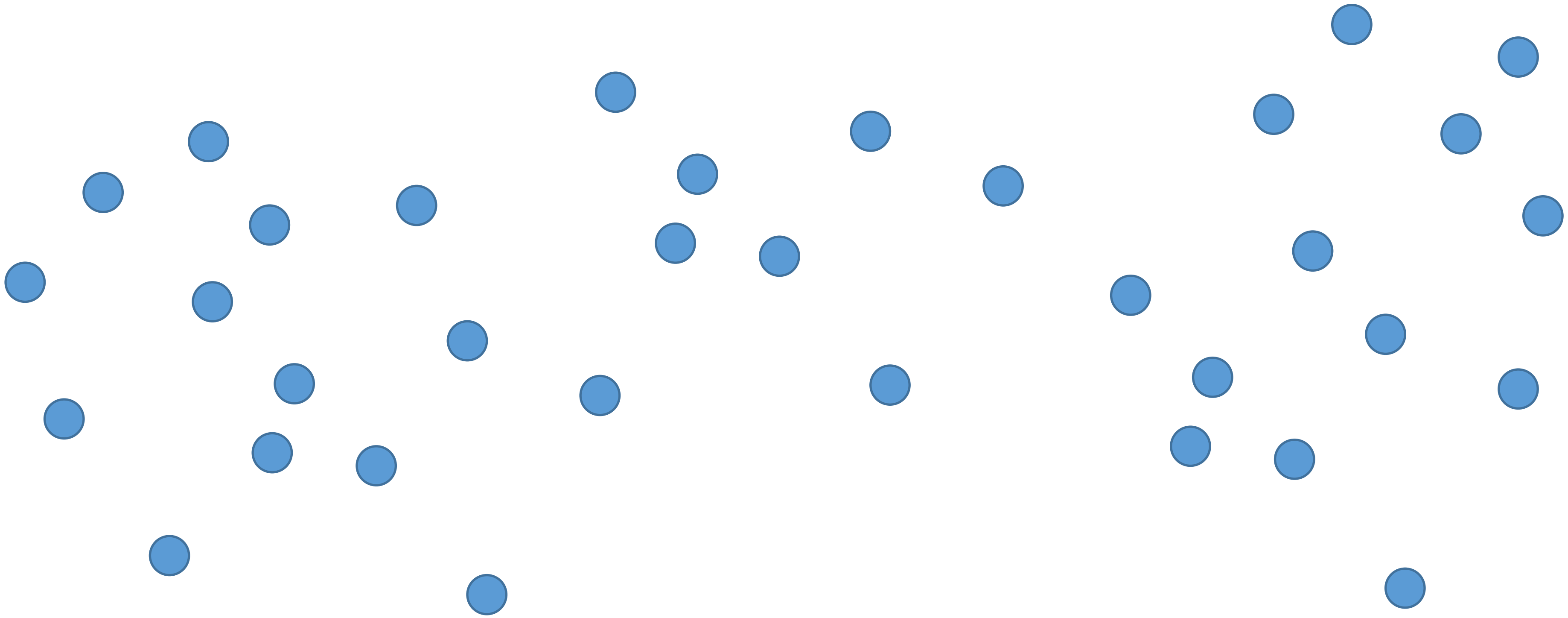
1. We are given a similarity (or dissimilarity) value for all points
2. Similarities are symmetric

$d(p, q)$ is the similarity between points p and q

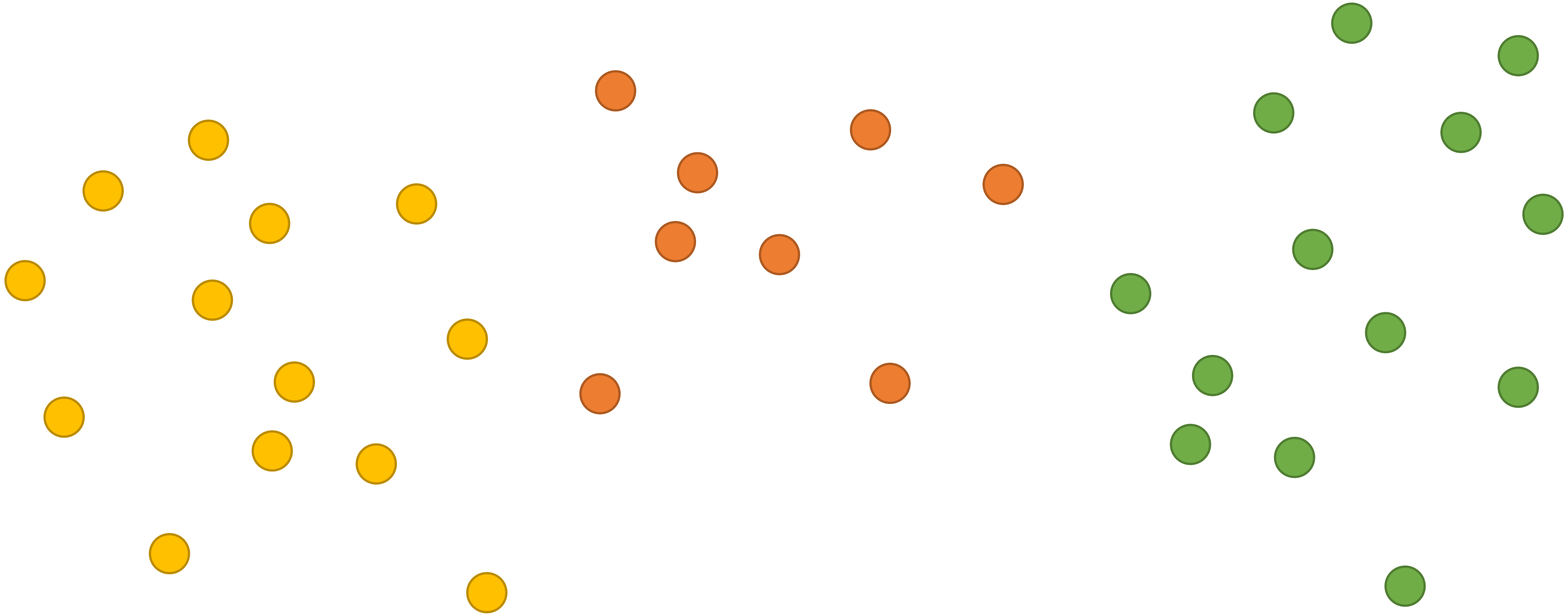
And $d(p, q) = d(q, p)$

Examples include Euclidean distance and edit distance

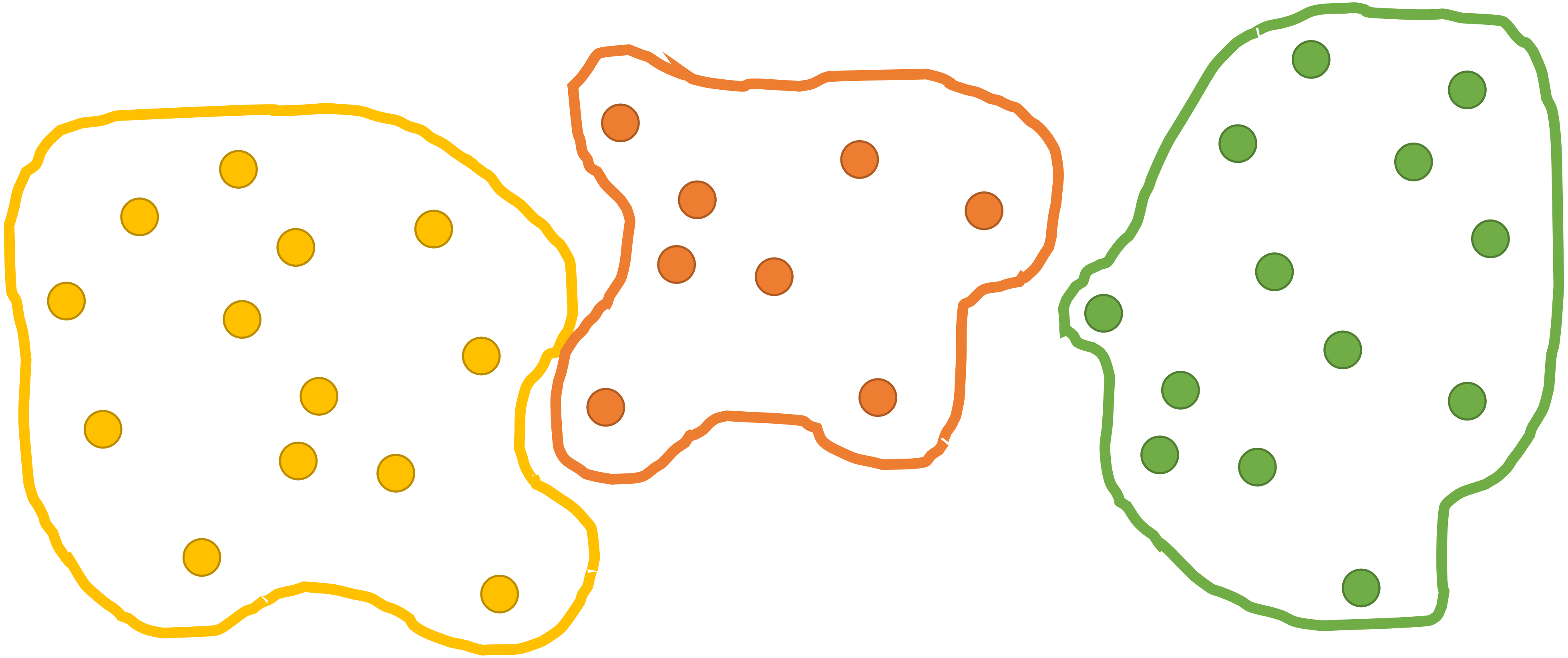
Goal: cluster ”nearby” points

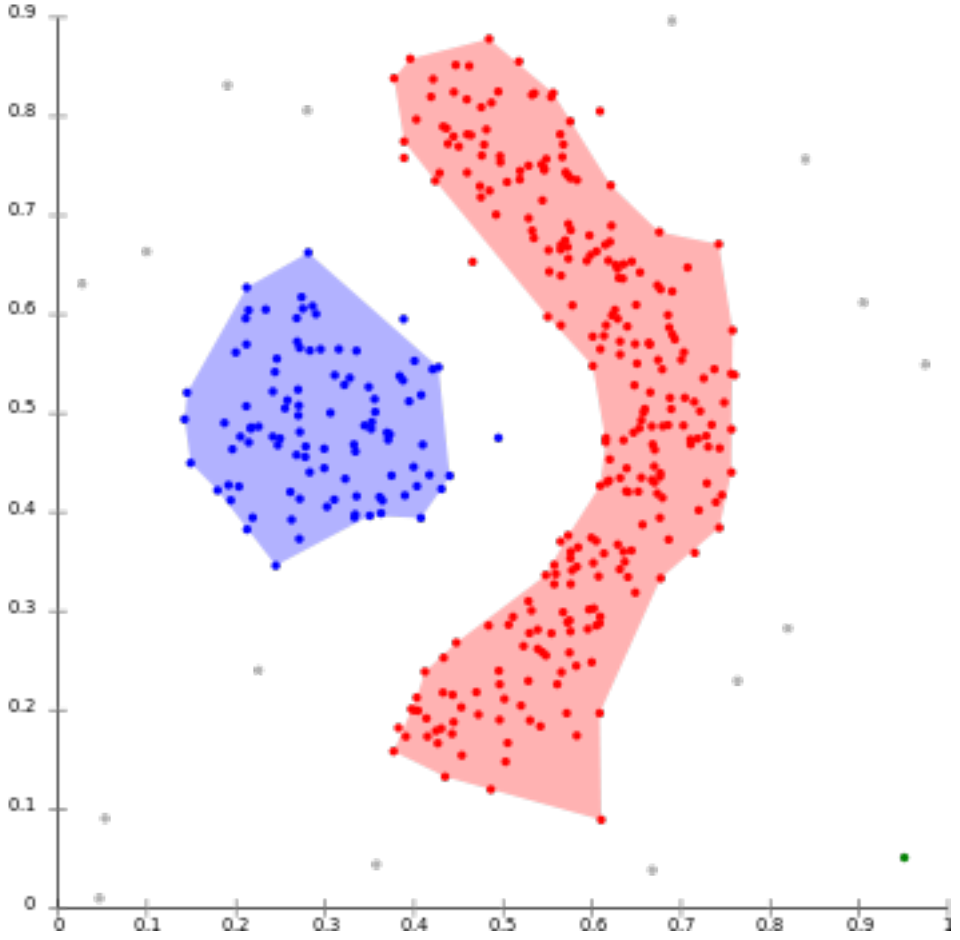


Goal: cluster ”nearby” points



Goal: cluster ”nearby” points





Clustering Topics/Algorithms

- Related to data mining, statistical data analysis, machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

- Hierarchical clustering
- Centroid clustering (k-means!)
- Distribution Clustering
- Density Clustering

Max-Spacing K-Clustering

- We assume that we know a good value for k, where k is the number of clusters that we are going to form.
- k is **not** discovered completely automatically (pick a few values are try them out).

- Two p and q points are separated if they are in different clusters.
- Thus, points that are similar should not be separated.
- Spacing S for a set of k-clusters is given by:

$$S = \min_{for\ all\ separated\ p,q} d(p, q)$$

- Given the above definition, what does a relatively large value for S signify?

Max-Spacing K-Clustering

- Problem statement: given a distance measure d and a cluster count k, compute the k-clustering with a maximum spacing S.

- Let's solve this problem with a greedy approach.
- Greedy algorithm setup:
  - Ignore k (the number of clusters) we produce until the end
  - Start by putting every point into its own cluster
  - How do we make spacing larger each iteration?
  - What is our greedy choice?

Max-Spacing K-Clustering

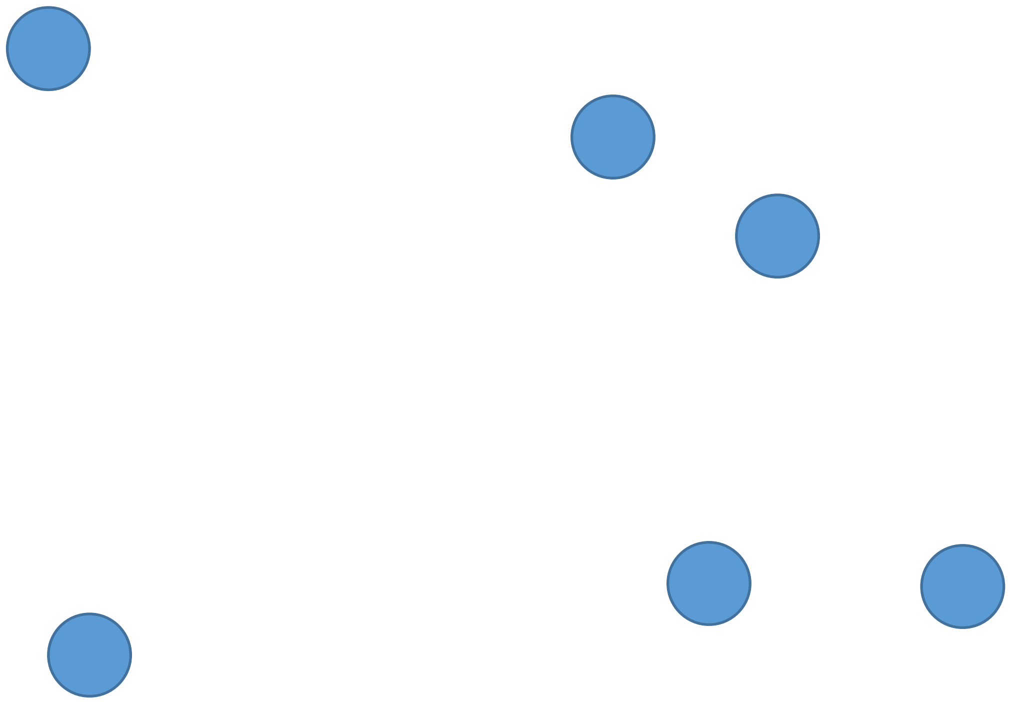
```
Put each point into its own cluster

Repeat until we have only k clusters
    let p, q = closest pair of separated points
    merge the clusters containing p and q
```

This is the operation that determines spacing

Max-Spacing K-Clustering

k = 3

`Put each point into its own cluster`

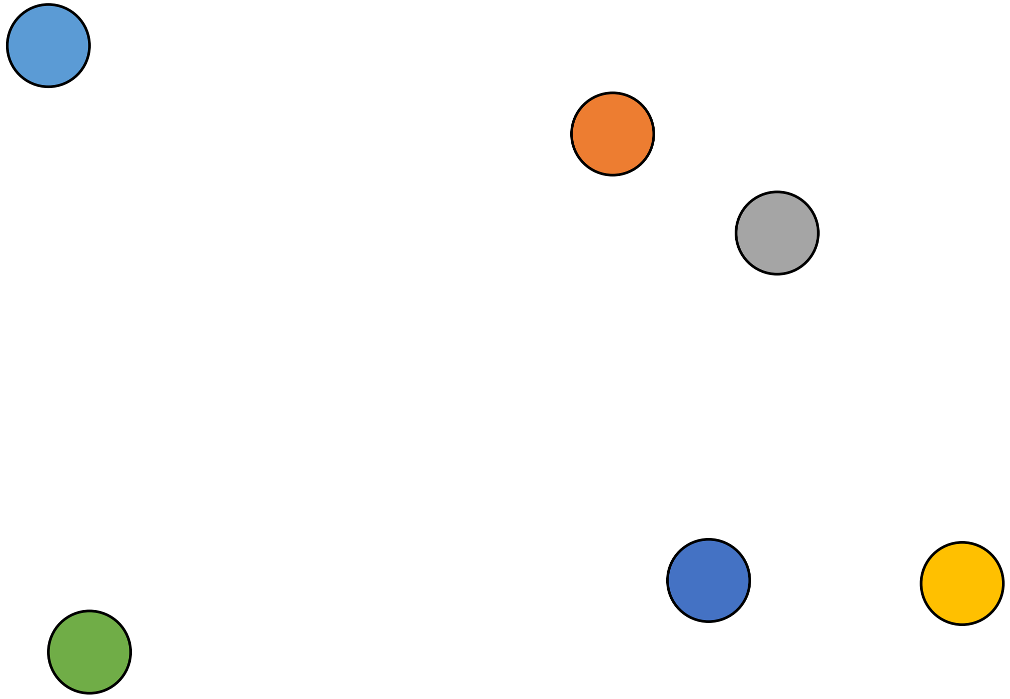


Max-Spacing K-Clustering

k = 3

Put each point into its own cluster

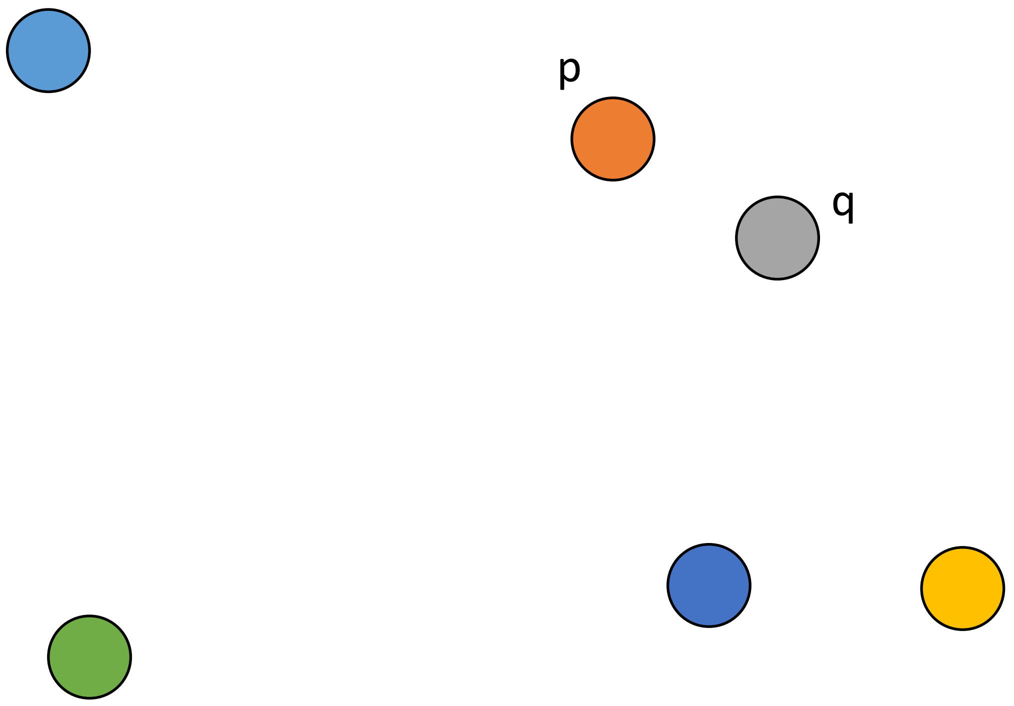
```
Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```



Max-Spacing K-Clustering

k = 3

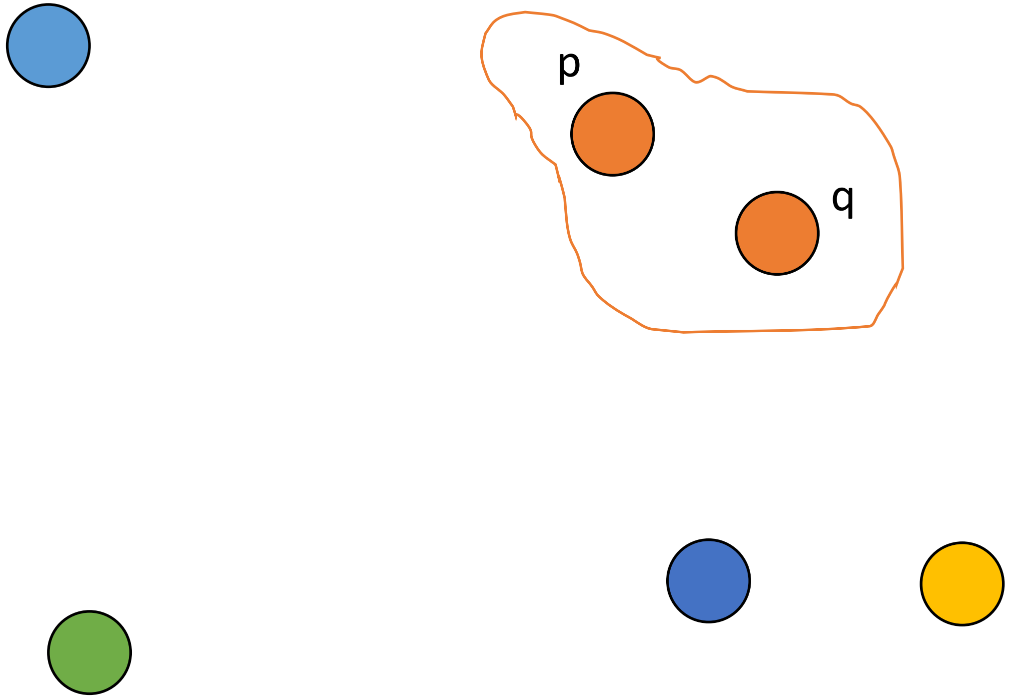
```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```



Max-Spacing K-Clustering

k = 3

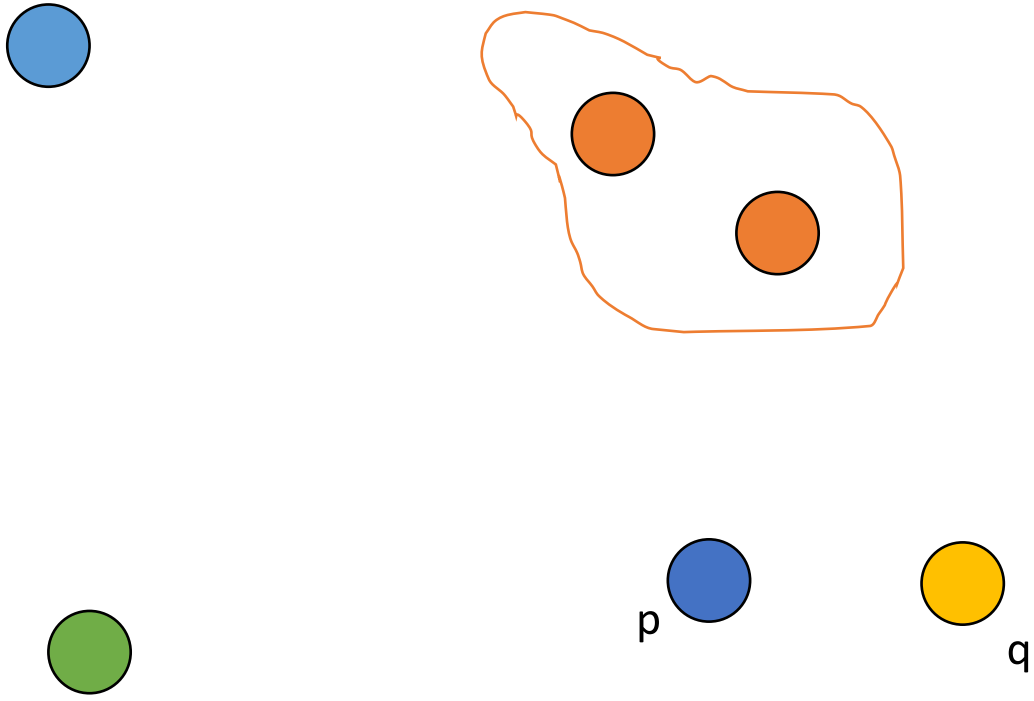
```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```



Max-Spacing K-Clustering

k = 3

```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```



Max-Spacing K-Clustering

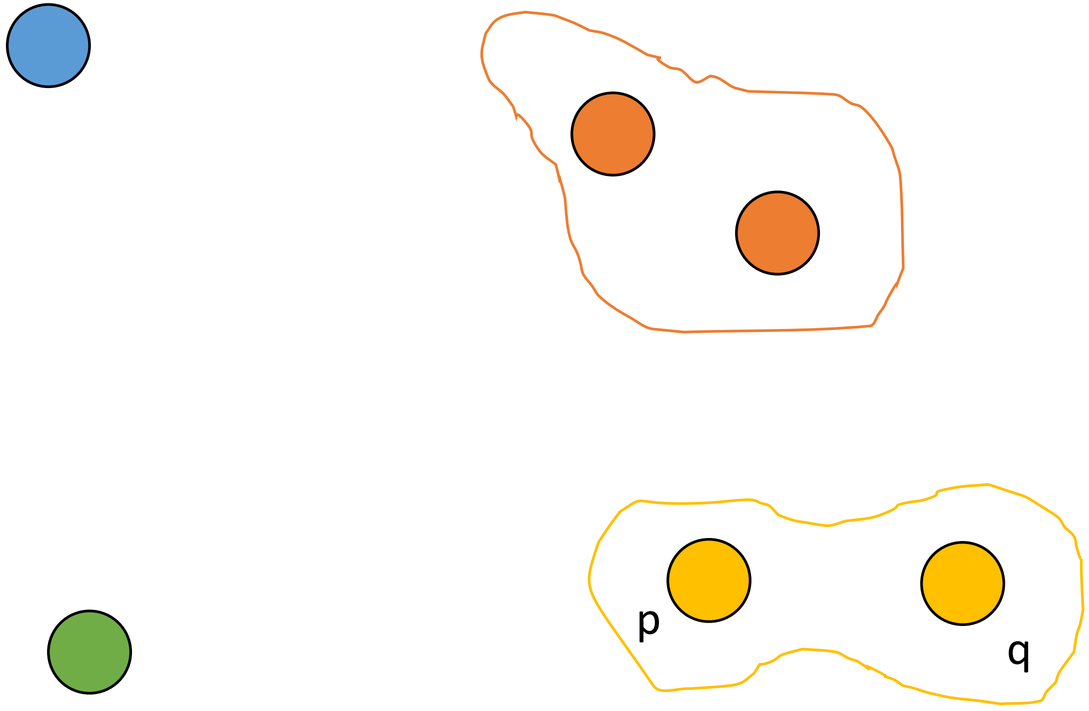
k = 3

Put each point into its own cluster

Repeat until we have only k clusters

p, q = closest pair of separated points

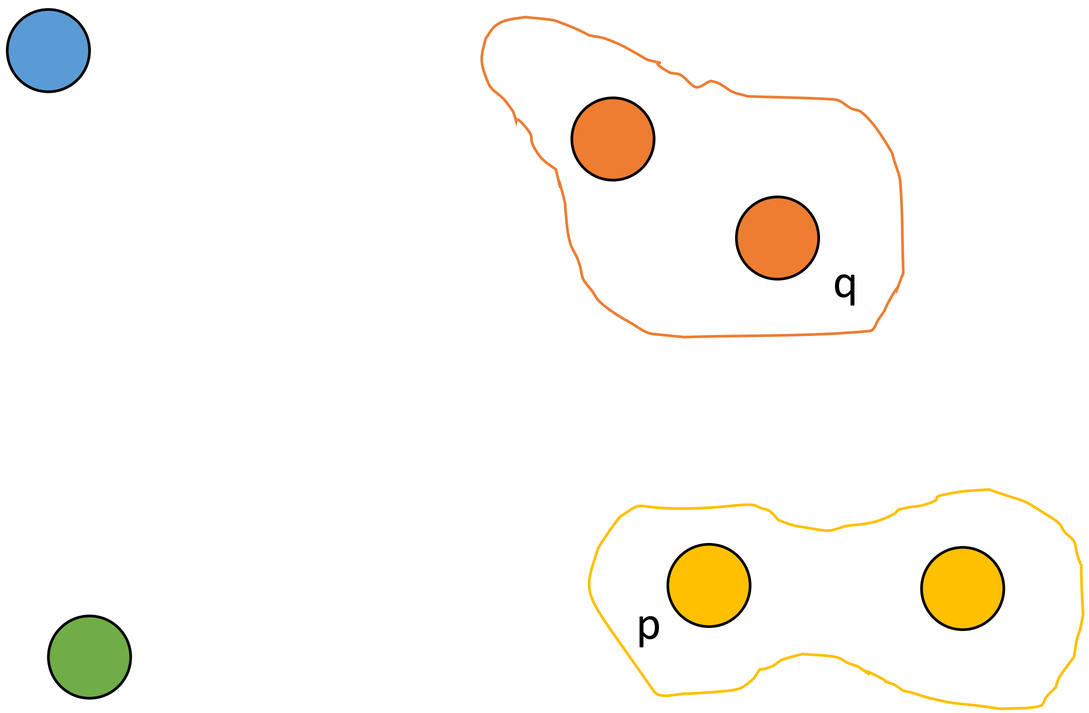
merge the clusters containing p and q



k = 3

Max-Spacing K-Clustering

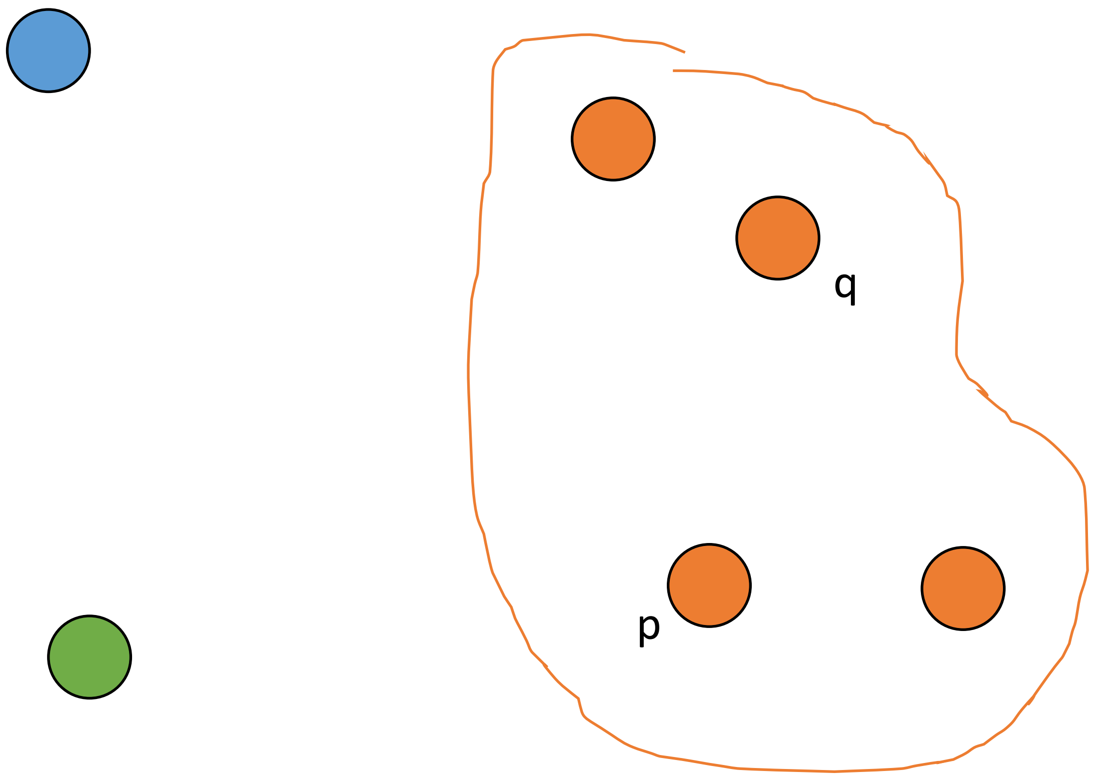
```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```



Max-Spacing K-Clustering

k = 3

```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```



Exercise Question 1

Does this algorithm look familiar?

- This procedure is nearly identical to Kruskal’s Algorithm for MST

Kruskals

```
Sort E by edge cost
T = empty
Each vertex into disjoint set

Repeat until only 1 set:
   u, v = next cheapest edge
   if Find(u) = Find(v)
      Union sets
```

Max-Spacing k-Clustering

```
Sort point pairs by d
C = empty
Each point into own cluster

Repeat until only k clusters:
   p, q = next closest points
   if p and q are separated
      Merge clusters
```

Does this algorithm look familiar?

- This procedure is nearly identical to Kruskal's Algorithm for MST

- What are the vertices?
- What are the edge costs?
- How many edges are there?
  - This gives us a "complete" graph.

- Using Kruskal's algorithm for cluster is called single link clustering.

Proof

Theorem: single-link clustering finds the max-spacing k-clustering of a set of points.

- Although we are using Kruskal's algorithm, the objective has changed.
- So, we **cannot** use the proof from before.

Exchange Argument

- Let C1, …, Ck be the k clusters computed by the greedy algorithm
- Let S be the spacing of these k clusters
- Let C1', …, Ck' be any other k clusters, with spacing S'

Exercise Question 2

- To prove our theorem, we need to show that S' ≤ S

Proof of Single-Link Clustering

- Note: it would be bad to find a case where S’ > S

- Case 1 (edge case): C1’, …, Ck’ are just a renaming C1, …, Ck
- In which case, S’ = S and we are done with this case

- Case 2: We can find a pair of points a and b such that:
  - a and b are in the same greedy cluster Ci
  - a and b are in different clusters Ca’, Cb’

Exchange

Proof of Single-Link Clustering

We have two cases to consider:

<u>Case 2a</u>: in the greedy algorithm, points a and b are **<u>directly</u>** merged at some point

<u>Case 2b</u>: in the greedy algorithm, points a and b are **<u>indirectly</u>** merged at some point

Proof of Single-Link Clustering

<u>Case 2a</u>: in the greedy algorithm, points a and b are **<u>directly</u>** merged at some point

- How does d(a, b) relate to S?

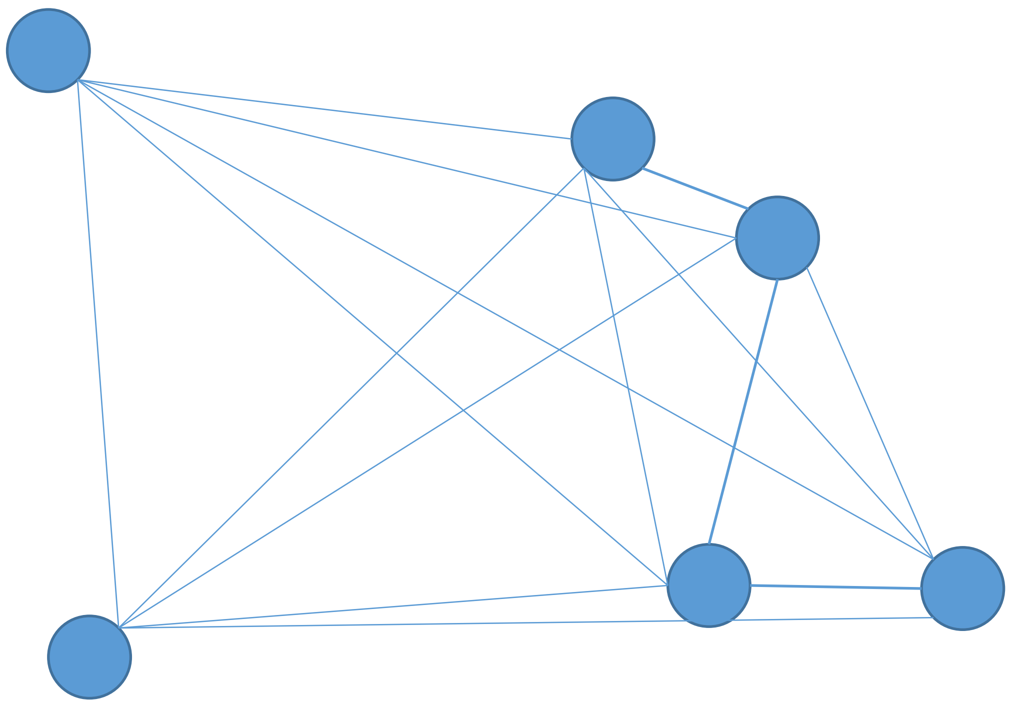
Max-Spacing K-Clustering

k = 3

S = ?

```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```

$$S = \min_{for\ all\ separated\ p,q} d(p, q)$$

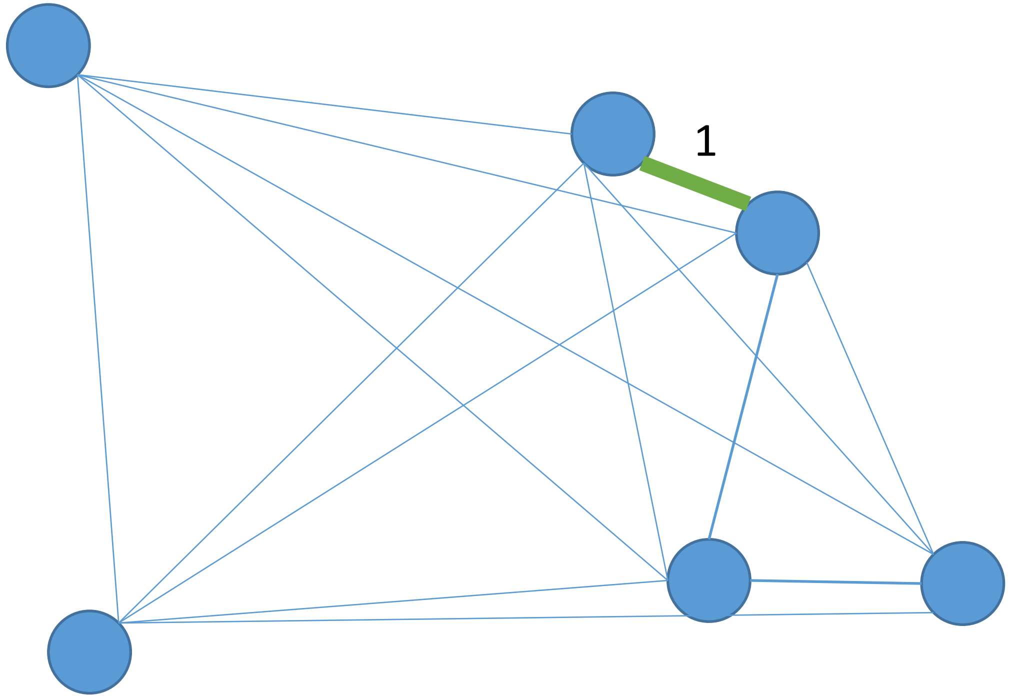


Max-Spacing K-Clustering

k = 3

S = 1

```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```



$$S = \min_{for\ all\ separated\ p,q} d(p, q)$$

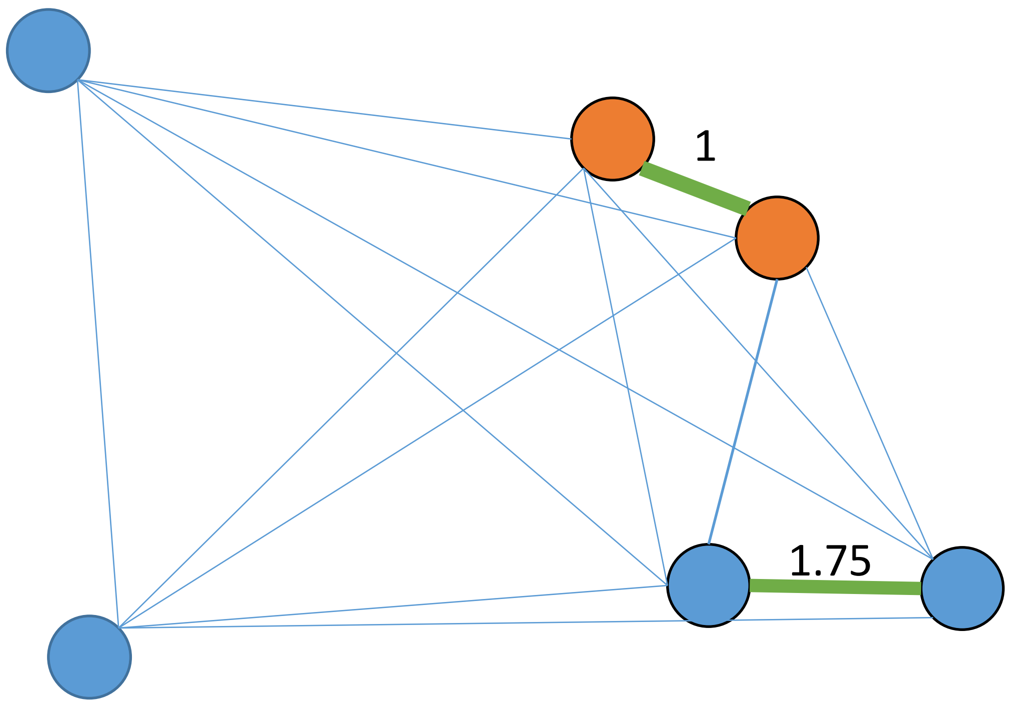
Max-Spacing K-Clustering

k = 3

S = 1.75

```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```

$$S = \min_{for\ all\ separated\ p,q} d(p,q)$$



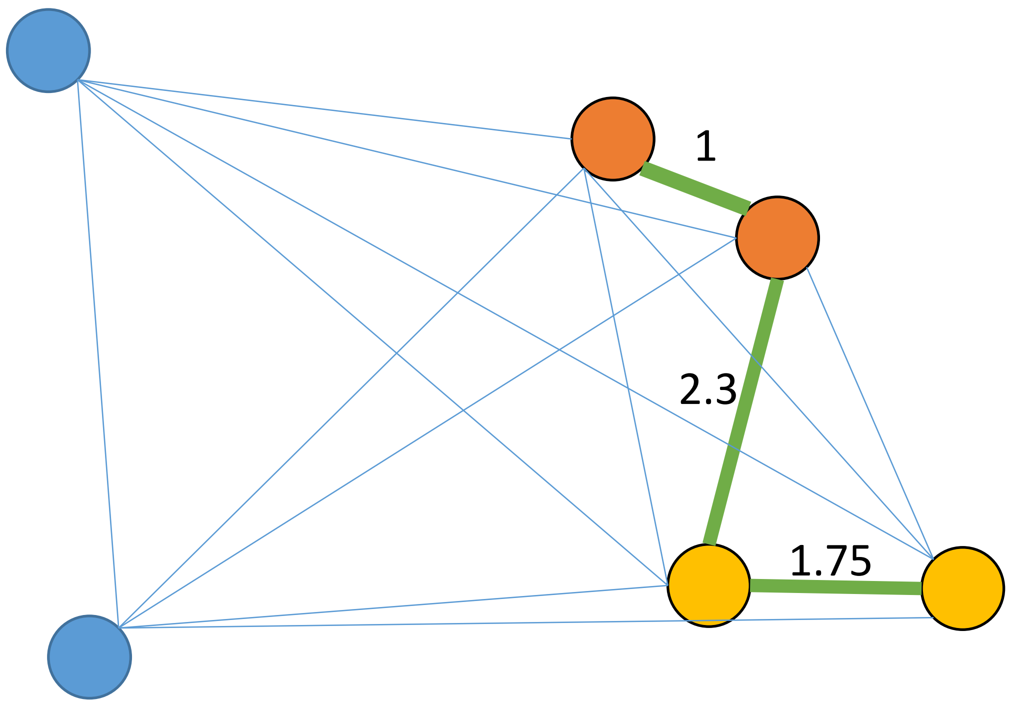
Max-Spacing K-Clustering

k = 3

S = 2.3

```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```

$$S = \min_{for\ all\ separated\ p,q} d(p, q)$$



Max-Spacing K-Clustering

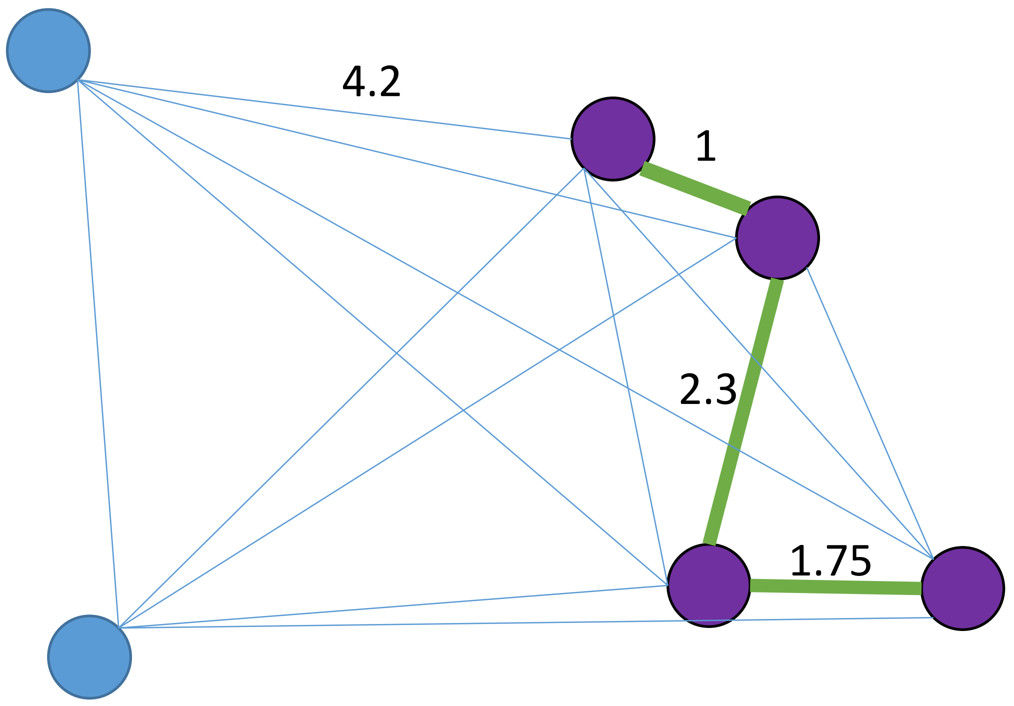
k = 3

S = 4.2

```
Put each point into its own cluster

Repeat until we have only k clusters
    p, q = closest pair of separated points
    merge the clusters containing p and q
```

$$S = \min_{for\ all\ separated\ p,q} d(p,q)$$

$$d(a,b) \leq S = 4.2$$



Proof of Single-Link Clustering

Case 2a: in the greedy algorithm, points a and b are **directly** merged at some point

- How does d(a, b) relate to S?

- If two points a and b are directly merged, then d(a, b) ≤ S
- Additionally, the distance between any two merged points only goes up (or stays the same) after each iteration

Proof of Single-Link Clustering

Case 2a: in the greedy algorithm, points a and b are **directly** merged at some point

- How does d(a, b) relate to S?

- If two points a and b are directly merged, then d(a, b) ≤ S
- Additionally, the distance between any two merged points only goes up (or stays the same) after each iteration

- So, we have that S' ≤ d(a, b) ≤ S → S' ≤ S

To prove our theorem, we need to show that S' ≤ S

Proof of Single-Link Clustering

We have two cases to consider:

Case 2a: in the greedy algorithm, points a and b are **directly** merged at some point

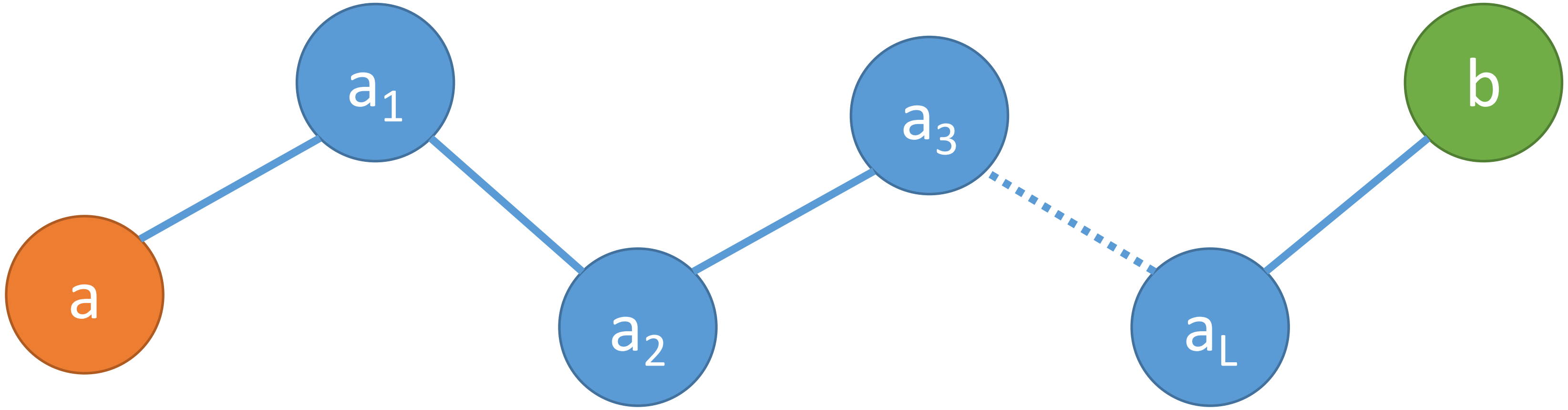
Case 2b: in the greedy algorithm, points a and b are **indirectly** merged at some point

Proof of Single-Link Clustering

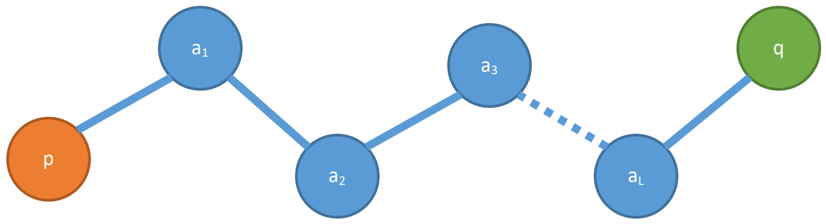
<u>Case 2b</u>: in the greedy algorithm, points a and b are **<u>indirectly</u>** merged at some point

- How does d(a, b) relate to S?
- Lines denote direct merges
- All points are in the same cluster in the end

$$d(a, b) \nless S = 4.2$$



Proof of Single-Link Clustering



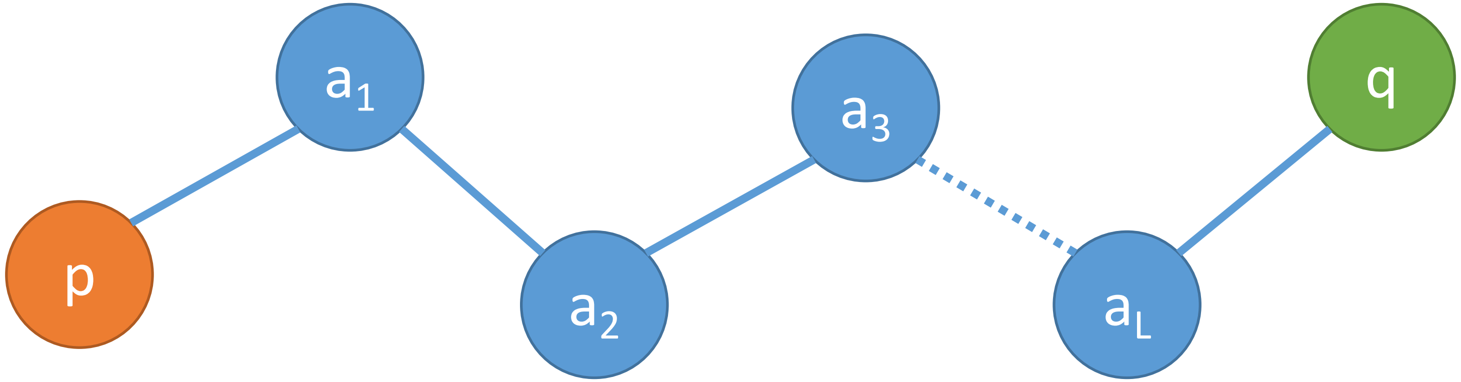
Case 2b: in the greedy algorithm, points a and b are **indirectly** merged at some point

> Case 2: We can find a pair of points a and b such that:
> a and b are in the same greedy cluster Ci
> a and b are in different clusters Ca', Cb'

- Let <a, a1, …, aL, b> be the path of direct merges connecting a and b
- In the non-greedy solution, since a is in Ca' and b is in Cb' there must be some consecutive pair where aj is in Ca' and aj+1 is in Cb'
- Thus S' ≤ d(aj, aj+1) ≤ S → S' ≤ S

Proof of Single-Link Clustering

- So, we have proved that under all circumstances, S is the biggest possible spacing for the points
- Thus, the greedy (Kruskal's-based) algorithm is optimal and correct



Preview for Dynamic Programming

How many ways can you return amount A using n kinds of coins?

All the ways returning amount A using all but the first kinds of coins (using the other (n – 1) kinds of coins)

+

All the ways returning amount (A – d) using n kinds of coins, where d is the denomination for the first kind of coin

Does this seem like a “hard” problem?