Northeastern University, Khoury College of Computer Science

# CS 6220 Data Mining | Assignment 4

Due: March 1, 2023(100 points)

Yiwei Cheng
sbchengyiwei
cheng.yiw@northeastern.edu

## K-Means

### Question 1

**[10 pts] Question 1a.) Here is the implementation of the k-means algorithm in Python on Colab:**

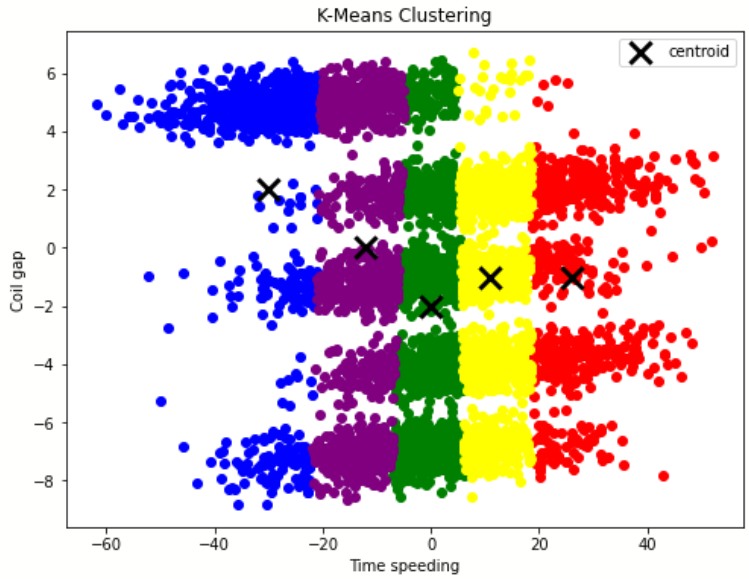
```
import numpy as np


def kmeans(data, k, initial_centroids):
 # initialize centroids to the given initial values
 centroids = initial_centroids
 # loop for a maximum of 100 iterations
 for i in range(100):
   # assign each data point to the closest centroid
   distances = np.zeros((len(data), len(centroids)))
   for i in range(len(data)):
       for j in range(len(centroids)):
           distances[i][j] = np.sqrt(np.sum((data[i] - centroids[j]) **
2))

   labels = np.argmin(distances, axis=1)

   # update the centroids
   for j in range(k):
     centroids[j] = np.mean(data[labels == j], axis=0)

 return centroids, labels
```

```
# initialize the centroids
initial_centroids = np.array([[10, 10], [-10, 10], [2, 2], [3, 3], [-3,
-3]])
scaler = StandardScaler()
scaled_centroids = scaler.fit_transform(initial_centroids)

# run the k-means algorithm
centroids, labels = kmeans(data, 5, initial_centroids)
```

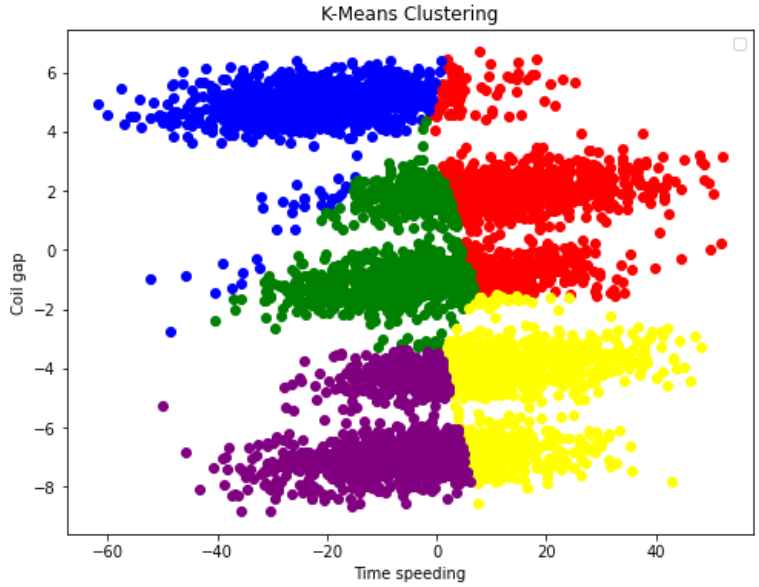
**[5 pts] Question 1b.) Scatter the results**

```
import matplotlib.pylab as plt

# visualize the results
fig = plt.figure(figsize=(8, 6))
colors = ['red', 'blue', 'green', 'yellow', 'purple']
for i in range(5):
 indices = np.where(labels == i)
 plt.scatter(data[indices, 0], data[indices, 1], c=colors[i])
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200,
linewidths=3, color='k', label='centroid')
plt.legend()
plt.xlabel('Time speeding')
plt.ylabel('Coil gap')
plt.title('K-Means Clustering')
plt.show()

```

The plot shows as follows, we can see that the scale is not correct.



Scaling the data is an important preprocessing step that helps to normalize the data and bring all the features to the same scale. This ensures that each feature contributes equally to the clustering process. **After scaling the data** and run another time of the k-means algorithm on it, we got:




**[5 pts] Question 1c.) Why is k = 5 a logical choice for this dataset? After plotting your resulting clusters, what do you notice? Did it cluster very well? Is there an initialization that would make it cluster well?**

It is based on the underlying patterns of this dataset that suggest 5 initialization clusters are good for the clustering effect. We can also see from the scatter plot that the dataset obviously scatters in 5 clusters.

The cluster was not very well, even after we scale the data and centroid. After trying some new initializations, I found they would not make the cluster result better. We should use other methods to deal with the scale.

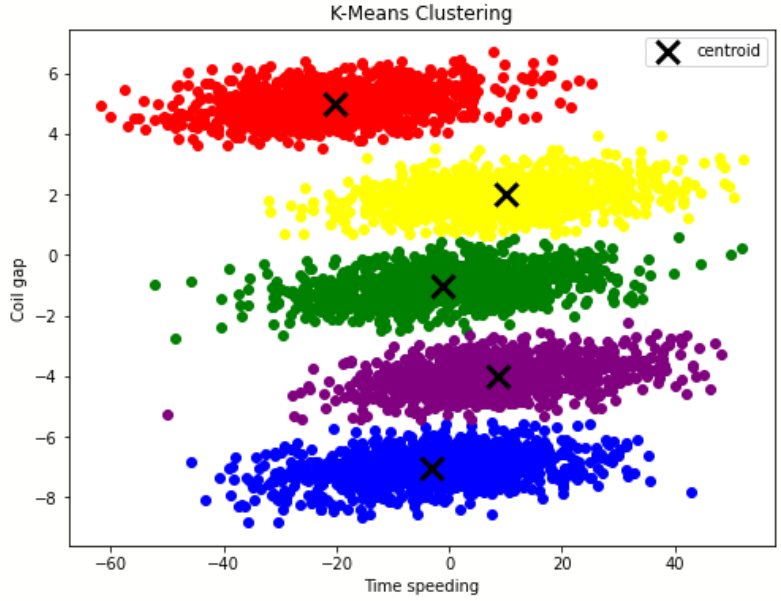
# Question 2

**[15 pts] Question 2a.)**

```
# p is a known relationship through Ford's quality control analysis
P = np.array([[10, 0.5], [-10, 0.25]])
factor = np.linalg.inv(P.T @ P)

# Mahalanobis Distance
def mahalanobis_distance(x, y, factor):
    return (x - y).T @ factor @ (x - y)

def k_means(data, k, centroids):
    n_iterations = 1000
    for i in range(n_iterations):
        distances = np.zeros((data.shape[0], k))
        for j in range(k):
```

```
            distances[:, j] = [mahalanobis_distance(data[m],
centroids[j], factor) for m in range(data.shape[0])]
        labels = np.argmin(distances, axis=1)
        new_centroids = np.array([data[labels == j].mean(axis=0) for j
in range(k)])
        if np.allclose(new_centroids, centroids):
            break
        centroids = new_centroids
    return centroids, labels

initial_centroids = np.array([[10, 10], [-10, -10], [2, 2], [3, 3],
[-3, -3]])
centroids, labels = k_means(data, 5, initial_centroids)
fig = plt.figure(figsize=(8, 6))
# visualize the results
colors = ['red', 'blue', 'green', 'yellow', 'purple']
for i in range(5):
 indices = np.where(labels == i)
 plt.scatter(data[indices, 0], data[indices, 1], c=colors[i])
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200,
linewidths=3, color='k', label='centroid')
plt.legend()
plt.xlabel('Time speeding')
plt.ylabel('Coil gap')
plt.title('K-Means Clustering')
plt.show()
```

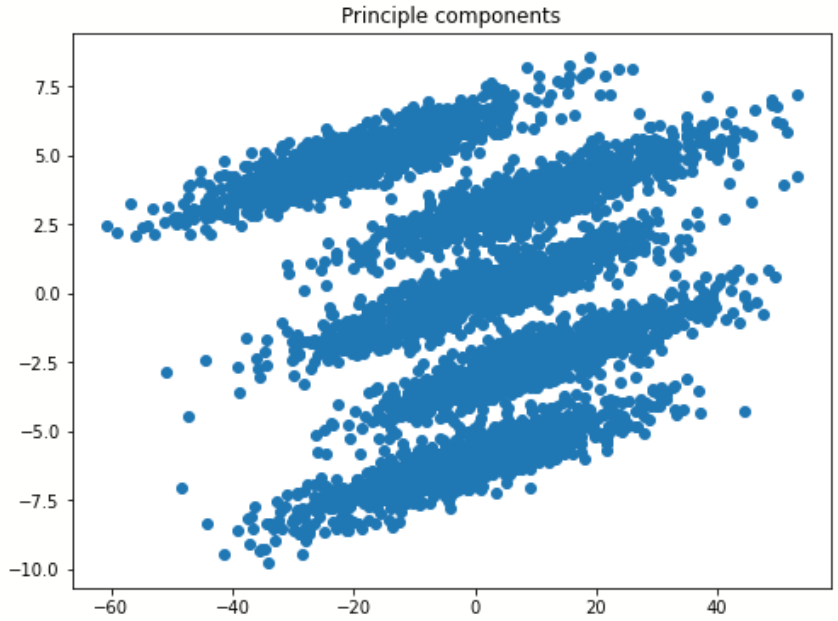
The plot shows as follows:




From this plot, we can see that the clustering result is much better. It clusters into five different groups.

**[5 pts] Question 2b.)**

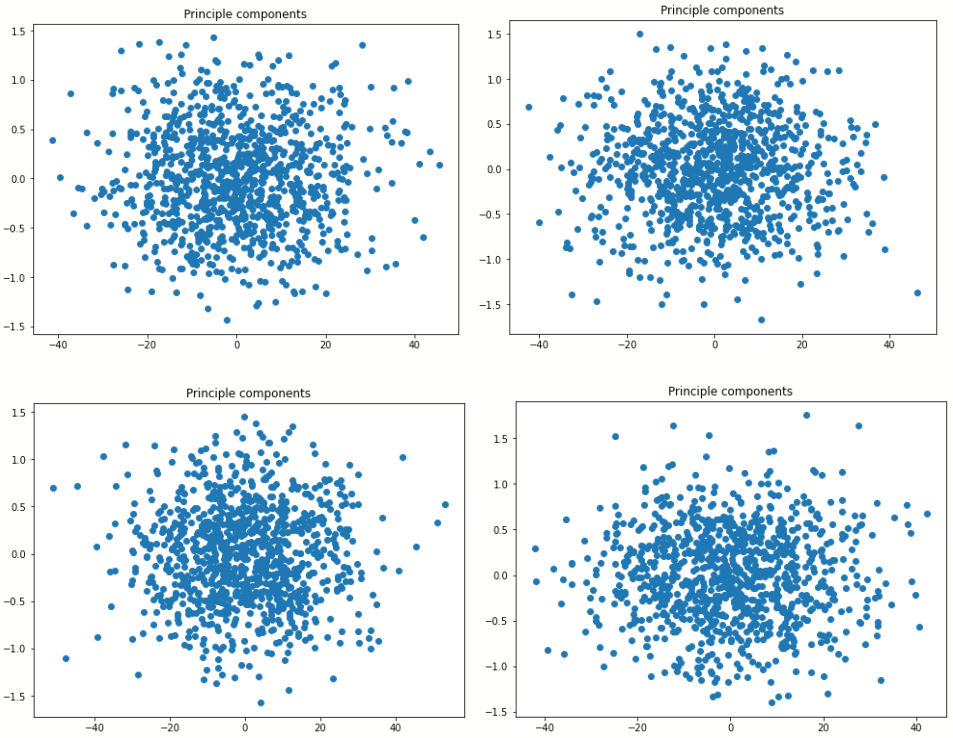
After calculating the principle components of the aggregate data and scattering it we got:

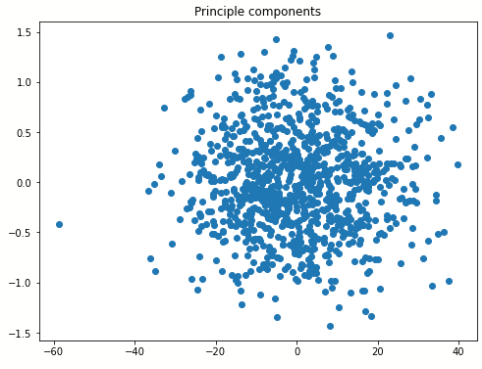



（The result is printed in the colab)

**[5 pts] Question 2c.)**

After calculating the principle components of each cluster's data and scattering it we got:





(The result is printed in the colab)

We can see that they are not as same as the aggregate data and each other. But the principle components of each cluster are similar.

**[15 pts] Question 2d.)**

```
P = np.array([[10, 0.5], [-10, 0.25]])

# Calculate the eigenvectors and eigenvalues of PT
eig_vals, eig_vecs = np.linalg.eig(P.T)

# Construct the diagonal matrix of eigenvalues
D = np.diag(eig_vals)

# Construct the matrix of eigenvectors
V = eig_vecs

# Print the diagonal matrix of eigenvalues and eigenvalues
print("Diagonal matrix of eigenvalues:\n", D)
print("Eigenvalues:\n", V )

# Print the product of V and D
print("P' = V @ D:\n", V @ D)

# Reconstruct the matrix P using the eigendecomposition
P_reconstructed = V @ D @ V.T

# Print the reconstructed matrix P
print("Reconstructed matrix P:\n", P_reconstructed)
```

```
Diagonal matrix of eigenvalues:
 [[9.45693086 0.        ]
 [0.         0.79306914]]
Eigenvalues:
 [[0.99852863 0.7356769 ]
 [0.05422701 0.67733264]]
P' = V @ D:
 [[9.44301625 0.58344264]
 [0.51282107 0.53717161]]
```

```
Reconstructed matrix P:
 [[9.85834739 0.90725126]
 [0.90725126 0.39165261]]
```

The product of the eigenvectors and eigenvalues in this case is represented by the matrix V @ D, which yields a matrix that is used to transform the data into a new coordinate system defined by the eigenvectors. The transpose of the eigenvectors matrix V.T is then used to rotate the data back into its original coordinate system. Thus, the product of the eigenvectors and eigenvalues represents a way to transform and reconstruct the data based on its underlying structure.

# Market Basket Analysis and Algorithms

## Question 3

**[10 pts] Question 3a.)**

To get the candidate 4-itemsets using the $F_{k-1} \times F_1$ merging strategy, we need to join each itemset in $F_3$ with each individual item in the dataset ($F_1$). If the resulting itemset has size 4 and all of its 3-item subsets are frequent, then it is a candidate 4-itemset.

So we need to join each of the 7 3-itemsets in F3 with each of the 5 items in the $F_1$:

{1, 2, 3} joined with {1}, {2}, {3}, {4}, and {5} yields:
{1, 2, 3, 1}, {1, 2, 3, 2}, {1, 2, 3, 4}, {1, 2, 3, 3}and {1, 2, 3, 5}
{1, 2, 3, 4} and {1, 2, 3, 5} are candidate 4-itemsets.

{1, 2, 4} joined with {1}, {2}, {3}, {4}, and {5} yields:
{1, 2, 4, 1}, {1, 2, 4, 2}, {1, 2, 4, 3}, {1, 2, 4, 4}, {1, 2, 4, 5}
{1, 2, 4, 5} is a candidate 4-itemset.

{1, 2, 5} joined with {1}, {2}, {3}, {4}, and {5} yields:
{1, 2, 5, 1}, {1, 2, 5, 2}, {1, 2, 5, 3}, {1, 2, 5, 4}, {1, 2, 5, 5}
None.

{1, 3, 4} joined with {1}, {2}, {3}, {4}, and {5} yields:
{1, 3, 4, 1}, {1, 3, 4, 2}, {1, 3, 4, 3},{1, 3, 4, 4}, {1, 3, 4, 5}
{1, 3, 4, 5} is a candidate 4-itemset.

{2, 3, 4} joined with {1}, {2}, {3}, {4}, and {5} yields:
{2, 3, 4, 1}, {2, 3, 4, 2}, {2, 3, 4, 3}, {2, 3, 4, 4}, {2, 3, 4, 5}
{2, 3, 4, 5} is a candidate 4-itemset.

{2, 3, 5} joined with {1}, {2}, {3}, {4}, and {5} yields:
{2, 3, 5, 1}, {2, 3, 5, 2}, {2, 3, 5, 3}, {2, 3, 5, 4}, {2, 3, 5, 5}
None.

In that case, we can get all candidate 4-itemsets as follows:
{1, 2, 3, 4} {1, 2, 3, 5} {1, 2, 4, 5} {1, 3, 4, 5} {2, 3, 4, 5}

**[10 pts] Question 3b.)**
To get the candidate 4-itemsets using the $F_{k-1} \times F_{k-1}$ strategy, we need to find $F_{k-1}$ which has the same k –1 items. So we could merge {1, 2, 3} and {1, 2, 4}, and get {1, 2, 3, 4}.
Merge {1, 2, 3} and {1, 2, 4}, and get {1, 2, 3, 4}.
Merge {1, 2, 4} and {1, 2, 5}, and get {1, 2, 4, 5}.
Merge {1, 2, 3} and {1, 2, 5}, and get {1, 2, 3, 5}.
Merge { 2, 3, 4} and { 2, 3, 5}, and get {2, 3, 4, 5}.

Using this method, we can get all candidate 4-itemsets as follows:
{1, 2, 3, 4} {1, 2, 3, 5} {1, 2, 4, 5} {2, 3, 4, 5}

**[5 pts] Question 3c.)**
**We don' t know the minimum support threshold times given the question prompt.**
Suppose we have a dataset with items {1}, {2}, {3}, {4}, and {5}, and the minimum support threshold is 2.
If the frequent 3-itemsets are {1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 4}, {2, 3, 4}, {2, 3, 5}, {3, 4, 5}, then the candidate 4-itemsets that survive the candidate pruning step can be found by generating all possible 4-itemsets from the frequent 3-itemsets and checking if all their 3-item subsets are frequent.
In general, the candidate pruning step in the Apriori algorithm removes any candidate k-itemset if any of its (k-1)-item subsets is not frequent, where frequent means that the itemset appears in the dataset at least the minimum support threshold times.
In the question 3a, we can got all candidate 4-itemsets as follows: {1, 2, 3, 4} {1, 2, 3, 5} {1, 2, 4, 5} {1, 3, 4, 5} {2, 3, 4, 5}. We can check each candidate against its 3-item subsets to see if they are all frequent.
{1, 3, 4, 5} has 3-item subsets {1, 3, 4}, {1, 3, 5}, {1, 4, 5}, {3, 4, 5}. According to the merge rule, we could merge {1, 3, 4}, {1, 3, 5} to get it, but we didn't have it in the frequent 3-itemsets. So it should be pruned off.
Therefore, the candidate 4-itemsets that survive the candidate pruning step could be {1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, and {2, 3, 4, 5}, but we still need to look into the whole table and see whether their support is larger than the threshold.

# Question 4

**[3 pts] Question 4a.)**
The maximum number of association rules that can be extracted from this data can be calculated by using the formula:
Maximum number of rules = 3^(number of distinct items) - 2^(number of distinct items + 1) + 1
To find the number of distinct items, we can scan the entire dataset and make a list of all the unique items. In this case, the distinct items are Beer, Diapers, Milk, Bread, Butter, Cookies, and Eggs.
Therefore, the maximum number of association rules that can be extracted from this data is

3^7 - 2^(7+1) + 1 = 2186 - 256 + 1 = 1931

**[3 pts] Question 4b.)**
To calculate the confidence of the rule {Milk, Diapers} ⇒ {Butter}, we need to use the following formula:
Confidence({Milk, Diapers} ⇒ {Butter}) = support({Milk, Diapers, Butter}) / support({Milk, Diapers})

We can calculate the support of the itemset {Milk, Diapers, Butter} by counting the number of transactions that contain all three items, which is 2: transaction 2, 7. The support of the itemset {Milk, Diapers} can be calculated by counting the number of transactions that contain both items, which is 4: transactions 2, 3, 5, and 7.
Therefore, the confidence of the rule {Milk, Diapers} ⇒ {Butter} is:
Confidence({Milk, Diapers} ⇒ {Butter}) = support({Milk, Diapers, Butter}) / support({Milk, Diapers}) = 2 / 4 = 0.5
So, the confidence of the rule {Milk, Diapers} ⇒ {Butter} is 0.5.

**[3 pts] Question 4c.)**
The support of the rule {Milk, Diapers} ⇒ {Butter} is the percentage of transactions that contain both {Milk, Diapers} and {Butter} among all transactions. We can calculate it using the following
support({Milk, Diapers, Butter}) / total number of transactions
To find the support of {Milk, Diapers, Butter}, we need to count the number of transactions in which all three items appear. From the given table, we can see that this itemset appears in transactions 2, 7. Therefore, the support of {Milk, Diapers, Butter} is 2.
The total number of transactions in the table is 10. Therefore, the support of the rule {Milk, Diapers} ⇒ {Butter} is:
support({Milk, Diapers, Butter}) / total number of transactions = 2 / 10 = 0.2
So the support of the rule {Milk, Diapers} ⇒ {Butter} is 0.2, and the absolute support of the rule {Milk, Diapers} ⇒ {Butter} is 2

**[3 pts] Question 4d.)**
True. If {a,b,c,d} is a frequent itemset, it means that it appears frequently in the transactions of the dataset, which implies that all its subsets must also appear at least as frequently. In particular, {a,b}, being a subset of {a,b,c,d}, must also appear at least as frequently as {a,b,c,d} in the transactions of the dataset.
Therefore, {a,b} is always a frequent itemset if {a,b,c,d} is a frequent itemset. However, it is possible that {a,b} appears more frequently in the transactions than {a,b,c,d}, in which case {a,b} would be a more frequent itemset than {a,b,c,d}.

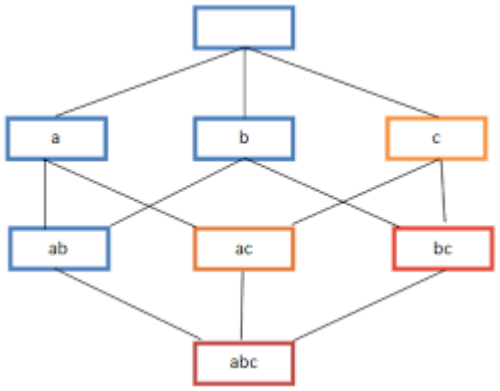
**[3 pts] Question 4e.)**
Not necessarily. Just because {a,b}, {b,c}, and {a,c} are frequent itemsets, it does not necessarily mean that their combination {a,b,c} is also frequent.

In general, the frequency of an itemset can be affected by the presence or absence of other items in the dataset. It is possible that {a,b,c} appears infrequently or not at all in the transactions of the dataset, even though {a,b}, {b,c}, and {a,c} are frequent.
Therefore, the statement is false. The frequent appearance of the individual itemsets does not guarantee that their combination is frequent. Whether or not {a,b,c} is frequent would depend on the specific dataset and the minimum support threshold used.

**[3 pts] Question 4f.)**
False. We cannot determine the support of {b} from the given information. If we have exactly 20 {a, b} and 30 {b, c}, the support of {b} would be 50 which is larger than 30.

**[3 pts] Question 4g.)**
False. If a dataset has 5 distinct items, then the number of possible size-2 itemsets is 5 choose 2, which is equal to 10.

**[3 pts] Question 4h.)**
The itemset lattice for this set of unique items would have the following itemsets:
{a}
{b}
{c}
{a, b}
{a, c}
{b, c}
{a, b, c}
{} (empty set)
The empty set is always included in the itemset lattice. Each node in the lattice represents an itemset, and there is a directed edge between itemsets A and B if A is a proper subset of B (i.e., all items in A are also in B, but B has at least one additional item).




In this case, the lattice would have 8 nodes and 12 edges.