National Technical University of Athens
School of Electrical and Computer Engineering
Inter-Faculty Postgraduate Studies Programme
Data Science and Machine Learning

First Set of Exercises

The first assignment written in partial fulfillment of the requirements for the completion of the Algorithmic Data Science core course of the Data Science & Machine Learning post-graduate studies programme.

Instructors
Prof. A. Pagourtzis
Dr. D. Souliou

Written by
[redacted]

April 26, 2022

Table of Contents

1 Frequent Itemsets Exercises

1.1 Exercise 6.3.1, MMDS

(a) Denoting the support of an itemset or rule X by $s(X)$, the following table (Table 1.1) shows the support of each 1-itemset, as well as whether it is considered to be frequent or not, based on the fact that the given support threshold is $t = 4$.

X	$s(X)$	Frequent
$\{1\}$	4	✓
$\{2\}$	6	✓
$\{3\}$	8	✓
$\{4\}$	8	✓
$\{5\}$	6	✓
$\{6\}$	4	✓

Table 1.1: Support of all 1-itemsets.

Evidently, all 1-itemsets are frequent. Moving on, the procedure is repeated for all 2-itemsets of the form $\{i, j\}$. In addition, the quantity $(i \cdot j) \mod 11$ is also calculated (to be utilized in the following question) and all results are presented in Table 1.2.

$\{i, j\}$	$s(\{i, j\})$	Frequent	$(i \cdot j) \mod 11$
$\{1, 2\}$	2	×	2
$\{1, 3\}$	3	×	3
$\{1, 4\}$	2	×	4
$\{1, 5\}$	1	×	5
$\{1, 6\}$	0	×	6
$\{2, 3\}$	3	×	6
$\{2, 4\}$	4	✓	8
$\{2, 5\}$	2	×	10
$\{2, 6\}$	1	×	1
$\{3, 4\}$	4	✓	1
$\{3, 5\}$	4	✓	4
$\{3, 6\}$	2	×	7
$\{4, 5\}$	3	×	9
$\{4, 6\}$	3	×	2
$\{5, 6\}$	2	×	8

Table 1.2: Support of all 2-itemsets.

It appears that only three 2-itemsets are frequent, namely $\{2, 4\}$, $\{3, 4\}$ and $\{3, 5\}$ and their support is exactly equal to the threshold. It may be noted at this point that, since the three frequent 2-itemsets contain a total of four unique items, there is no reason to count frequent 3-itemsets, as it can be known a-priori that none of them is frequent.

(b) The hash function $h(i,j) = (i \cdot j) \mod 11$, with $i \neq j$, maps all 2-itemsets $\{i,j\}$ into 10 distinct buckets, each of which can be denoted by the corresponding integer value of $h(i,j)$. The bucket where each 2-itemset is hashed can be seen in the fourth column of Table 1.2.

(c) Combining the second and the fourth column of Table 1.2, one can calculate the number of elements hashed into each bucket and therefore determine whether the bucket is frequent or not. The results are shown in Table 1.3.

Bucket	Number of elements	Frequent
1	5	✓
2	5	✓
3	3	×
4	6	✓
5	1	×
6	3	×
7	2	×
8	6	✓
9	3	×
10	2	×

Table 1.3: The number of elements in each bucket.

Out of the ten buckets, only buckets No. 1, 2, 4 and 8 are frequent and therefore the 1 bit is assigned to them during the construction of the bitmap. The 0 bit is assigned to the remaining six buckets.

(d) According to the PCY algorithm, only the pairs that are hashed into frequent buckets are considered for its second pass as candidate pairs. Joining Tables 1.2 and 1.3 on the Bucket column, one easily finds that the candidate pairs are $\{2,6\}$ and $\{3,4\}$ (from bucket 1), $\{1,2\}$ and $\{4,6\}$ (from bucket 2), $\{1,4\}$ and $\{3,5\}$ (from bucket 4), $\{2,4\}$ and $\{5,6\}$ (from bucket 8).

1.2 Exercise 6.3.2, MMDS

Since the first part of the Multistage algorithm is the same as the first pass of the PCY algorithm, one can continue from the end of the previous exercise. There, it was determined that a total of 4 buckets were frequent, corresponding to 8 unique candidate 2-itemsets. The new hash function, $\tilde{h}(i,j) = (i + j) \mod 9$, has to be applied to each of these unique 2-itemsets, in order to assign each of them to one out of eight possible buckets. The results are shown in Table 1.4 (left), which is the equivalent of Table 1.2 from the previous exercise, but now contains only 8 instead of 15 pairs of items.

The integer value of the hash function is assigned to each (new) bucket and the number of elements within the bucket is counted, in order to determine whether it is frequent or not. The results are presented in Table 1.4 (right). It becomes evident that only buckets No. 6, 7 and 8 are frequent, and therefore the number of candidate pairs is reduced from 8 to 4. The final candidate pairs are $\{2,4\}$ (from bucket 6), $\{3,4\}$ (from bucket 7), $\{2,6\}$ and $\{3,5\}$ (from bucket 8). The efficiency of the Multistage algorithm becomes apparent, since one additional hashing procedure reduces the total number of candidate pairs to half. Note that from the 4 candidate pairs, only $\{2,6\}$ is not in fact frequent, as can be seen from its support in Table 1.2.

$\{i, j\}$	$s(\{i, j\})$	$(i + j) \mod 9$
$\{2, 6\}$	1	8
$\{3, 4\}$	4	7
$\{1, 2\}$	2	3
$\{4, 6\}$	3	1
$\{1, 4\}$	2	5
$\{3, 5\}$	4	8
$\{2, 4\}$	4	6
$\{5, 6\}$	2	2

$\Longrightarrow$

Bucket	Number of elements	Frequent
1	3	×
2	2	×
3	2	×
4	0	×
5	2	×
6	4	✓
7	4	✓
8	5	✓

Table 1.4: Hashing of the candidate 2-itemsets acquired from the first pass of the PCY algorithm.

1.3 Exercise 6.4.1, MMDS

The maximal frequent itemsets are those for whom all immediate supersets are infrequent and they therefore constitute the positive border. Given the positive border, one can find the negative border, i.e. the infrequent itemsets for whom all immediate subsets are frequent. Fig. 1.1 depicts the border separating the frequent from the infrequent itemsets and shows the itemsets that constitute the positive and the negative border with yellow and blue backgrounds, respectively. The full powerset pow({A,B,C,D,E,F,G,H}) is not portrayed because it contains 255 itemsets (excluding the null) and would therefore require sizes larger than A4 (or really small fonts).

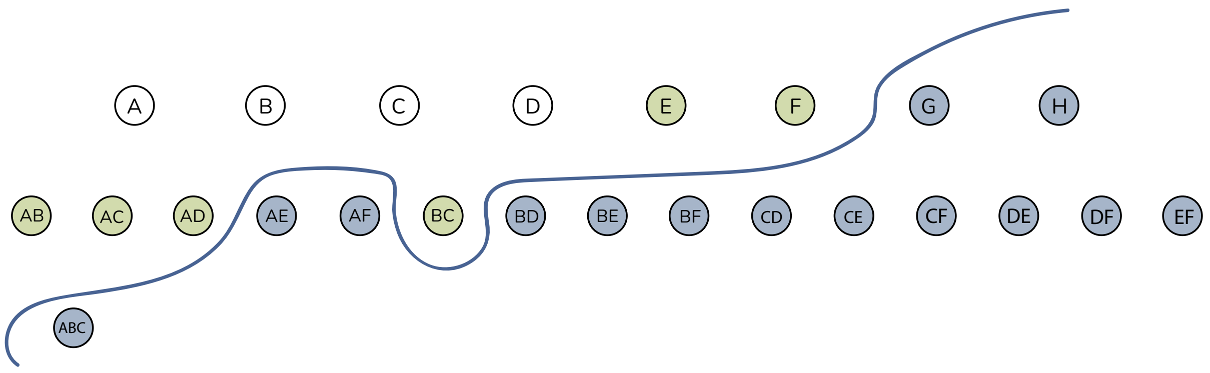

Figure 1.1: The border separating the frequent from the infrequent itemsets.

The reasoning behind this choice can be summarized by studying k-itemsets separately as follows.

<u>1-itemsets</u>: Since neither the 1-itemsets {G} and {H} nor any superset thereof is present in the positive border, {G} and {H} must belong to the negative border. All other 1-itemsets are frequent: the 2-itemsets {A,B}, {A,C} and {A,D} are frequent, thus the unique items that constitute them must also be frequent. Also, {E} and {F} belong to the positive border, so they are also frequent.

<u>2-itemsets</u>: Since {E} and {F} belong to the positive border, all supersets thereof must be infrequent. As a result, {E,F}, as well as any 2-itemset of the form {X,E} or {X,F} with X = A, B, C or D, must belong to the negative border, because E, F and X are frequent. On the other hand, {A,B}, {A,C}, {A,D} and {B,C} are frequent, therefore the only 2-itemsets left to examine are {B,D} and {C,D}. If any of these two is frequent, there must be at least one corresponding superset that is also frequent, since these do

not belong to the positive border. However, the supersets {A,B,D} and {B,C,D} cannot be frequent, as {A,B} and {B,C} are maximally frequent, therefore {B,D} is infrequent. Similarly, the supersets {A,C,D} and {B,C,D} cannot be frequent, as {A,C} and {B,C} are maximally frequent, therefore {C,D} is also infrequent. Finally, since {B,D} and {C,D} are infrequent, while {B}, {C} and {D} are frequent, one concludes that they belong to the negative border.

3-itemsets: Based on the a-priori algorithm, the only 3-itemset that can be constructed from frequent 2-itemsets is {A,B,C}. This set must be infrequent, as {A,B} (or, equivalently, {A,C} or {B,C}) belong to the positive border. Therefore, {A,B,C} belongs to the negative border.

Summarizing, the itemsets $\{G\}$, $\{H\}$, $\{A,E\}$, $\{A,F\}$, $\{B,D\}$, $\{B,E\}$, $\{B,F\}$, $\{C,D\}$, $\{C,E\}$, $\{C,F\}$, $\{D,E\}$, $\{D,F\}$, $\{E,F\}$ and $\{A,B,C\}$ constitute the negative border.

2 Market Basket Data

(a) In the context of the market basket data, the a-priori algorithm is based on the following observation: if an itemset is frequent, then all of its subsets must also be frequent, a property which is due to the anti-monotonicity of support. Inversely, if an itemset is infrequent, then all of its supersets are also infrequent. Based on this observation, the a-priori algorithm significantly speeds up the process of frequent itemset mining. In particular, consider a set of items Σ, with powerset $J = \text{pow}(\Sigma)$. A brute-force algorithm would count the support of each element in J in order to determine which ones are frequent and which ones are not. According to the a-priori algorithm, if any k-itemset is found to be infrequent, then all k'-itemsets that are its supersets ($k' > k$) will also be infrequent by default, with no counting required to reach this conclusion. One of the main disadvantages of the a-priori algorithm is that it requires q passes across the database, if q is the highest cardinality among the frequent itemsets. For example, supposing that $\{a,b,c,d,e\}$ are the unique items in one such database, if $\{a,b,d\}$ is the frequent itemset with the highest cardinality (which is 3), then the a-priori algorithm would require 3 passes across the database in order to converge.

(b) The terms "input" and "output" complexity both refer to the algorithm's time complexity[1]. The input complexity corresponds to the time complexity as far as the algorithm's input is concerned. For a database with an input of n unique items, the input complexity is $\mathcal{O}(2^n)$, as the cardinality of the corresponding powerset is $2^n - 1$ (excluding the null set) and in the worst case scenario all of the item-sets are frequent (which could happen, for example, in a database consisting of identical transactions). Strictly speaking, the input complexity is $\mathcal{O}(2^n \cdot P(n))$, where $P(n)$ is a polynomial of n, if one takes into account the required database passes, the average transaction width, etc. However, the term 2^n is enough by itself to determine that the algorithm's input complexity cannot be polynomial - it is always exponential. On the other hand, when it comes to a-priori's output complexity, it is equal to $\mathcal{O}\left(m \cdot n^2\right)$, where m is the number of frequent itemsets. While there are cases where $m \sim \mathcal{O}(2^n)$ and the output complexity is also exponential, more often than not this is not the case. Of course, this is not to say that the output complexity **is** polynomial, but rather that it **can be** in several cases.

(c) The discussion above is in fact algorithm-independent, so the same ideas apply in the case of the FP-Growth algorithm as well. Even though FP-Growth is faster compared to a-priori (this is demonstrated with practical experiments in Exercise 3) and also requires only 2 database passes in total, the facts that (i) the input complexity is bounded by the 2^n factor of the worst case scenario and (ii) the output can be polynomial in many practical cases, still hold.

[1] Thanks to Prof. A. Pagourtzis for an illuminating discussion via email correspondence on this subject.

(d) The unique items in the given database are $\{a, b, c, d\}$ and the elements of the corresponding powerset (excluding the null set) are shown in Fig. 2.1, where the notation $\{i, j\} \rightarrow ij$ is used for brevity.

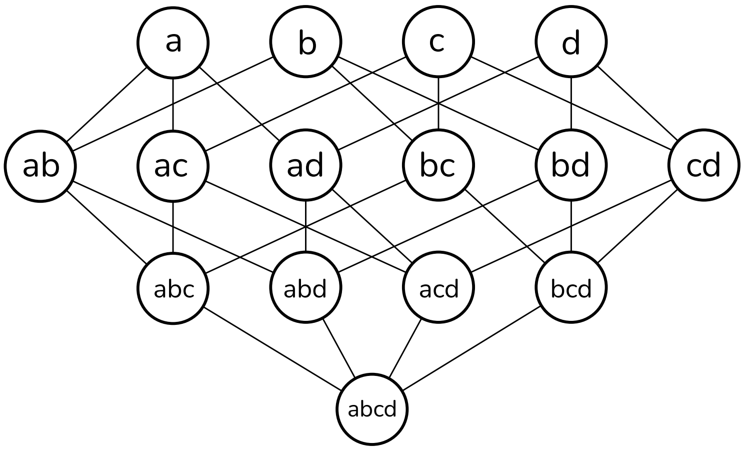

Figure 2.1: The elements of pow $(\{a, b, c, d\})$, excluding the null set.

In what follows, the support threshold is $t = 4$ and the support of an itemset or rule X is denoted by $s(X)$. As far as the a-priori algorithm is concerned, the support of all 1-itemsets is calculated for its first pass:

$$s(a) = 7 > t, \quad s(b) = 6 > t, \quad s(c) = 7 > t, \quad s(d) = 9 > t$$

Evidently, all 1-itemsets are frequent, so all 2-itemsets are candidates. For the second pass of the a-priori algorithm one finds:

$$s(ab) = 2 < t, \quad s(ac) = 4 = t, \quad s(ad) = 5 > t, \quad s(bc) = 3 < t, \quad s(bd) = 5 > t, \quad s(cd) = 4 = t$$

When it comes to 2-itemsets, *ab* and *bc* are not frequent, which means that the only candidate 3-itemset is *acd*. As a result, the third pass of the a-priori algorithm simply calculates $s(acd) = 2 < t$ and the algorithm is terminated. Summarizing, the frequent itemsets that the a-priori algorithm finds are *a*, *b*, *c*, *d*, *ac*, *ad*, *bd* and *cd*. The results can be seen in Fig. 2.2, where the borders of frequent and infrequent itemsets are yellow and blue, respectively.

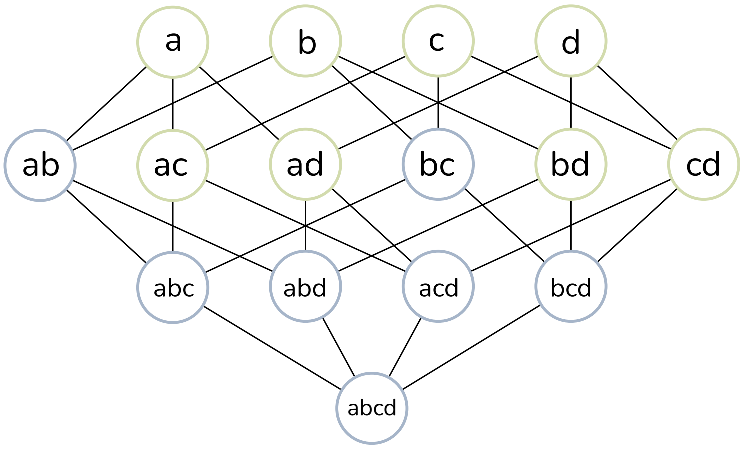

Figure 2.2: The itemsets split into frequent (yellow border) and infrequent (blue border).

For the FP-Growth algorithm, the first step is the calculation of the support for all 1-itemsets, which has already been done as part of the a-priori algorithm. Then, all infrequent items are removed from the transactions (no removals occur in this case) and each transaction is ordered by descending item support (i.e. d goes first, a and c follow and b goes last). Afterwards, the ordered transactions are depicted as a tree, as shown in Fig. 2.3.

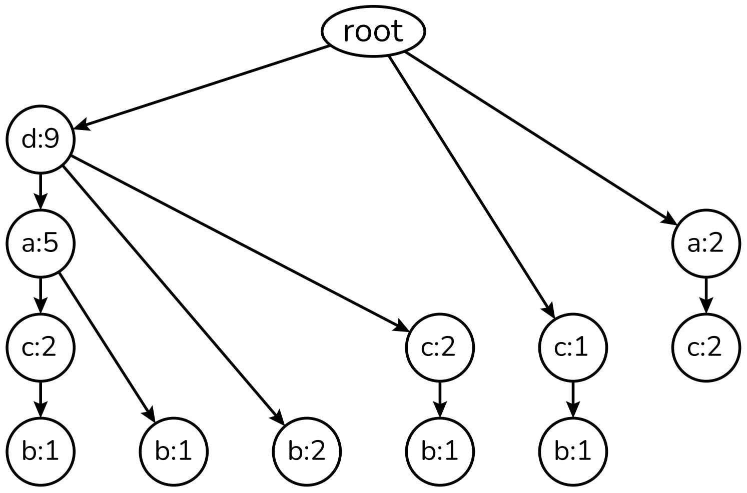

Figure 2.3: Tree of ordered transactions.

Finally, starting from the item with the lowest support (i.e. b), the tree is traversed backwards and the support of all itemsets containing each item is calculated.

In the present example, starting from b, one finds that $s(ab) = 2$, since b occurs once in two branches that also contain a, $s(bc) = 3$, since b occurs once in three branches that also contain c and $s(bd) = 5$, since b occurs twice in a branch that also contains d and once in three branches that also contain d. Moving on to c, after discarding b, it turns out that $s(ac) = 4$ and $s(cd) = 4$, as c occurs twice in two branches that also contain a and d, respectively. Finally, after discarding c as well, the only 2-itemset left is ad with support $s(ad) = 5$, since a occurs five times in a branch that also contains d. Summarizing, the frequent itemsets that the FP-Growth algorithm finds are a, b, c, d, ac, ad, bd and cd.

Closing this part of the study, it is worth making two remarks. Firstly, note that 3-itemsets were not mentioned above (in the FP-Growth algorithm), as their support is lower than the threshold. Secondly, the results of the two algorithms are obviously in complete agreement (this also serves as a sanity check).

(e) Moving on to Toivonen's algorithm, the proposed sample consists of the transactions shown below, in Table 2.1.

a	b	c	d
1	0	1	1
1	0	0	1
0	1	1	1

Table 2.1: Sample for Toivonen's algorithm.

Case 1: The first part of Toivonen's algorithm constists of the calculation of the sample's frequent itemsets. The a-priori algorithm is utilized for this purpose, so the results for the 1-itemsets are:

$$s(a) = 2 > t', \quad s(b) = 1 = t', \quad s(c) = 2 > t', \quad s(d) = 3 > t'$$

It appears that all 1-itemsets are frequent, given the threshold t', so all 2-itemsets are considered for the second pass:

$$s(ab) = 0 < t', \quad s(ac) = 1 = t', \quad s(ad) = 2 > t', \quad s(bc) = 1 = t', \quad s(bd) = 1 = t', \quad s(cd) = 2 > t'$$

Consequently, all 2-itemsets apart from *ab* are frequent. Since *ab* is infrequent, *abc* and *abd* must also be infrequent, so during the third pass only *acd* and *bcd* are considered. The results are $s(acd) = 1 = t'$ and $s(bcd) = 1 = t'$, which means that both of them turn out to be frequent in the given sample. Obviously, the single 4-itemset is not frequent, as 2 of its subsets are infrequent. Figure 2.4 depicts the results of the first part of Toivonen's algorithm, based on the sample of Table 2.1. The borders of the frequent and infrequent itemsets are yellow and blue, respectively, while the elements of the negative border (the itemset *ab*) are depicted with dashed borders.

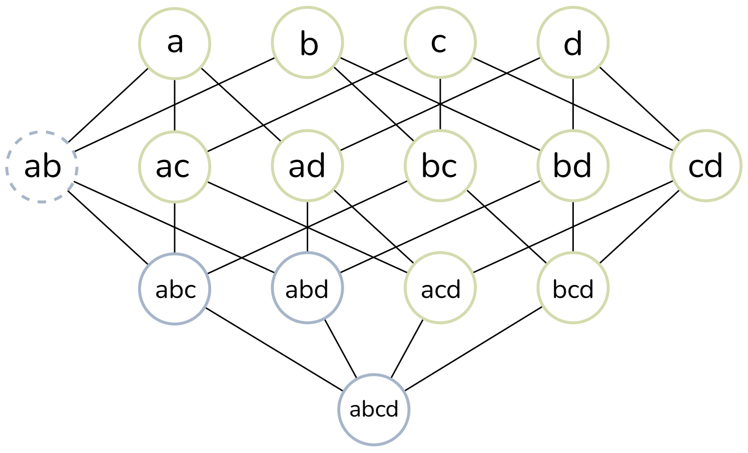

Figure 2.4: The results for the first part of Toivonen's algorithm.

For the second part of Toivonen's algorithm, the support of all frequent candidates, as well as the itemsets that constitute the negative border is calculated, using the original sample and support threshold. Of course, these have already been calculated in question **(d)**. As far as the negative border, i.e. the itemset *ab*, is concerned, it is known that $s(ab) = 2 < t$, which means that *ab* is infrequent. Since all constituents of the negative border are indeed infrequent, the implication is that no frequent itemsets were missed when choosing candidates. In other words, the itemsets that were deemed infrequent during the first part of Toivonen's algorithm are truly infrequent for the whole database. Indeed, the already known frequent itemsets, i.e. *a*, *b*, *c*, *d*, *ac*, *ad*, *bd* and *cd*, are among the candidates chosen from Toivonen's algorithm.

Case 2: The procedure is identical to the one shown in the first case. This time, however, the sample threshold is higher, so *b* turns out to be infrequent, as $s(b) = 1 < t''$. Since *b* is an infrequent 1-itemset in the sample, the first part of Toivonen's algorithm indicates that it must belong to the negative border, which is known to be untrue based on all previous calculations. A direct consequense of this is that Toivonen's algorithm will not be able to isolate the truly infrequent itemsets and will therefore fail to speed up the process of itemsets mining, as another run will be required (using either a larger/different sample, or a lower threshold).

3 Frequent Itemset Mining Algorithms

For the purposes of this exercise, two papers that were presented in the 2003 ICDM Workshop on Frequent Itemset Mining Implementations are studied. These are "*A Fast APRIORI implementation*", by F. Bodon [1] and "*Efficiently Using Prefix-trees in Mining Frequent Itemsets*", by G. Grahne & J. Zhu [2]. The reasoning behind this choice is that these papers share the same, interesting approach on the topic of frequent itemset mining: instead of trying to develop new algorithms in order to undertake the task, in hopes of achieving better performances, they focus on the implementation details of already existing algorithms and in fact the most basic among them - a-priori and FP-Growth. Bodon goes so far as to state that the effect of implementation can be more important than the selection of the algorithm and the experiments he presents seem to confirm his claim.

When it comes to the proposed implementations, Bodon presents a method to count itemsets' supports with tries, in a fashion similar to FP-Growth, and argues that taking advantage of the available space to store some additional information can significantly speed up the process of finding candidates. Furthermore, he utilizes hashing techniques in order to alter some (but not all) nodes of the tries into hash-tables, thus accelerating the search at nodes with many edges. His most important addition to the algorithm, however, is the so called "Brave" candidate generation. The APRIORI-BRAVE hybrid is a heuristic algorithm which diverges from the traditional a-priori algorithm during the last phases, where the number of candidates is small, but the database still needs to be fully scanned in order to determine their support. The idea behind APRIORI-BRAVE is the following: after determining the frequent k-itemsets, the generation of $(k + 1)$-itemset candidates occurs as in a-priori. However, the algorithm does not carry on with the support count, but checks if the memory need exceeds the maximal memory need. If not, $(k + 2)$-itemset candidates are generated and another memory check is performed in order to generate candidates from higher levels. This goes on as long as the memory need does not reach maximal memory need. In this way, apart from the $(k+1)$-itemset candidates, the candidates of the next phases are also collected and their support is counted in a single database pass.

Regarding Grahne and Zhu's paper, their basic observation (which serves as their motive) is that, while the FP-Growth algorithm shows an extremely good performance, there is still room for improvement when it comes to the traversing of the FP-trees. In fact, they found that about 80% of the algorithm's runtime consisted of this traversing, which is why they proposed the following idea: for each item i in the header of a conditional FP-tree T_X, two traversals of T_X are required in order to construct a new conditional FP-Tree, $T_{X\cup\{i\}}$ - one to count the frequent items in the conditional pattern base of $X \cup \{i\}$ and initialize the $T_{X\cup\{i\}}$ tree and one to construct it. However, the first scan of T_X can be omitted by constructing an array that holds all the required information for the initialization of $T_{X\cup\{i\}}$, during the construction of T_X. In this way, the process is sped up significantly at the cost of some additional space, which is unallocated from the memory once the FP-tree is, as the proposed array is an attribute of the tree itself.

Moving on to code implementations of the proposed techniques and algorithms, the code studied for Bodon's proposals is his own (version 2.4.9, 2005), written in C++ and available on his website[2]. The documentation is fairly simple, as only three parameters are mandatory: the file that contains the transactions in market-basket format, the output file where the frequent itemsets are written and the support threshold. Note that the support threshold is not given by an integer, as in the exercises studied herein, because it needs to be independent of the number of unique items or the total number of transactions, if one wishes to draw comparisons between different databases. Instead, it is given as a float which corresponds to a percentage of the total number of transactions.

As far as Grahne and Zhu's ideas are concerned, the code studied is not the one provided in

[2] http://www.cs.bme.hu/~bodon/en/apriori/

http://fimi.uantwerpen.be/, but rather a more recent and optimized version by Christian Borgelt, based on his paper "*An Implementation of the FP-growth Algorithm*", [3]. The newest version is 6.20, dated 2022, it is also written in C++ and is available on his website[3]. Since this implementation is being continuously developed during the past 17 years, it comes with a large number of options and approaches, from different variants of FP-Growth to additional evaluation measures for itemset support. For this reason, the documentation is somewhat more complex (see his website), although the algorithm is operational as long as one mandatory parameter is given: the file containing the transactions in market-basket format.

For the final part of this study, some basic experiments are presented, using both of these algorithms on different datasets. While using the original datasets and parameters mentioned in the papers by the authors in order to reproduce their results was the original plan, it quickly became evident that this had no practical value, since the technological advancements of the past two decades are more than significant when it comes to hardware. For example, Grahne and Zhu's experiments were performed on a 1 GHz Pentium III processor with 512 MB of RAM, while the results presented here correspond to experiments performed on an i7-10700KF Processor with 8 cores at 3.8 GHz and 32 GB of DDR4 RAM. As a result, this original idea was dismissed and a different approach was chosen, using additional datasets. The new idea was to compare the two algorithms as far as runtime is concerned, for different values of the support threshold.

In what follows, the results of this investigation are presented for each dataset separately. Note that all time results correspond to CPU times measured in seconds. Additionally, they correspond to the total time required for the convergence of the algorithms, without including the time required to write the results on the disk. Notice that the support thresholds chosen for the algorithms are much lower compared to the ones used in the aforementioned bibliography, since the available processing power is orders of magnitude higher compared to that of 2003.

3.1 The mushroom dataset

The mushroom dataset is one of the datasets prepared by Roberto Bayardo from the UCI datasets and PUMSB. It contains a total of 119 unique items and constists of 8124 transactions. It is one of the "lightest" datasets available in the exercise's repository in terms of number of transactions and frequent itemsets, which is the reason why it is chosen for the first experiment of this analysis. Both algorithms are executed for the support thresholds, t, shown in Table 3.1. As mentioned above, these thresholds are much lower compared to the ones mentioned in the bibliography, since modern day processors can perform tasks with large amounts of data exponentially faster compared to two decades ago.

t (%)	0.0005	0.001	0.002	0.004	0.008	0.01	0.02	0.05	0.1	0.5	1
A-priori (s)	1.65	1.58	1.53	1.53	1.53	1.51	1.51	1.40	1.37	0.94	0.72
FP-Growth (s)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

Table 3.1: Runtime results for the mushroom dataset.

The results indicate that, while both algorithms perform exceptionally well even for very low support thresholds (there is no mention of thresholds as low as 0.0005% in the given bibliography), FP-Growth is two orders of magnitude faster for the lowest threshold values and one order faster for the larger threshold values. In fact, as can be seen in the graph of Fig. 3.1, the runtime of FP-Growth does not seem to change as a function of the support threshold. The same can be told for the a-priori algorithm as well, but only for values $t \leq 0.1\%$. Note that the axes' scale in Fig. 3.1 is logarithmic.

[3] https://borgelt.net/fpgrowth.html

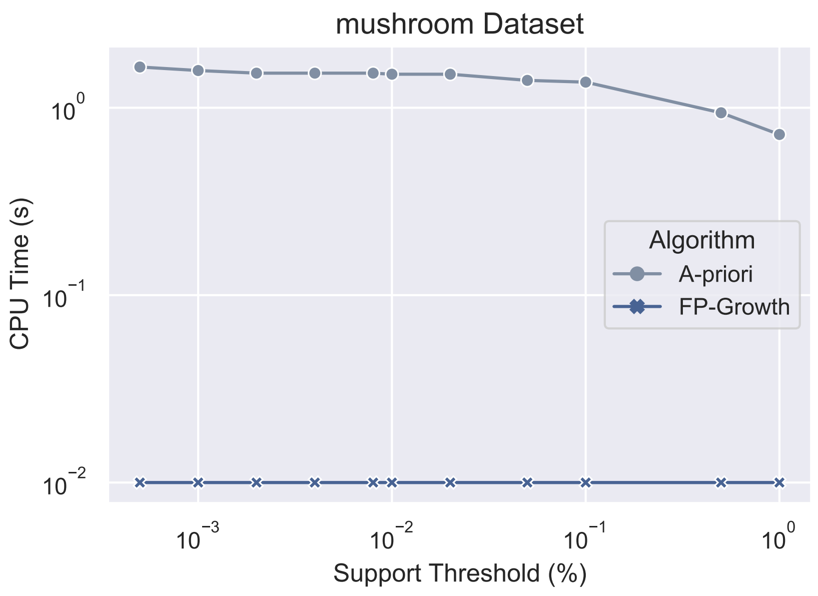

Figure 3.1: A-priori and FP-Growth runtime comparison for the mushroom dataset.

These results are a first indication that FP-Growth performs much faster compared to a-priori, however the algorithms' runtime does not vary significantly as a function of the support threshold. This could be due to the dataset's relatively small number of unique items combined with its small number of total transactions. For this reason, the same experiments are performed on a larger dataset.

3.2 The T10I4D100K dataset

The dataset chosen for this purpose is T10I4D100K, which corresponds to one of the datasets mentioned in the bibliography's papers. It contains 870 unique items and 100000 transactions, was generated from a generator by the IBM Almaden Quest research group and was used for many years in order to benchmark various algorithms for frequent itemset mining.

t (%)	0.0005	0.001	0.002	0.004	0.008	0.01	0.02	0.05	0.1	0.5	1
A-priori (s)	32.12	31.82	7.41	2.86	1.17	0.91	0.48	0.32	0.27	0.1	0.1
FP-Growth (s)	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06

Table 3.2: Runtime results for the T10I4D100K dataset.

The results for this dataset (see Table 3.2 or Fig. 3.2) seem to confirm the trend that was noticed for the FP-Growth algorithm, i.e. the fact that its runtime is not dependent upon the support threshold (for the studied threshold values). The increase that can be seen (from 0.1s to 0.6s) is probably due to the slightly larger time required to read the larger dataset (there are $\mathcal{O}\left(10^2\right)$ times more transactions than in the first dataset). The same cannot be told for the a-priori algorithm, the runtime of which seems to vary more than before: it takes nearly half a minute for the algorithm to converge for the lowest support threshold value and just a fraction of a second for the highest support threshold value. The obvious conclusion at this point is that larger databases indeed significantly affect the a-priori algorithm's runtime for certain support thresholds, while FP-Growth keeps outperforming a-priori with a constant $\mathcal{O}\left(10^{-2}\right) - \mathcal{O}\left(10^{-1}\right)$ seconds CPU runtime.

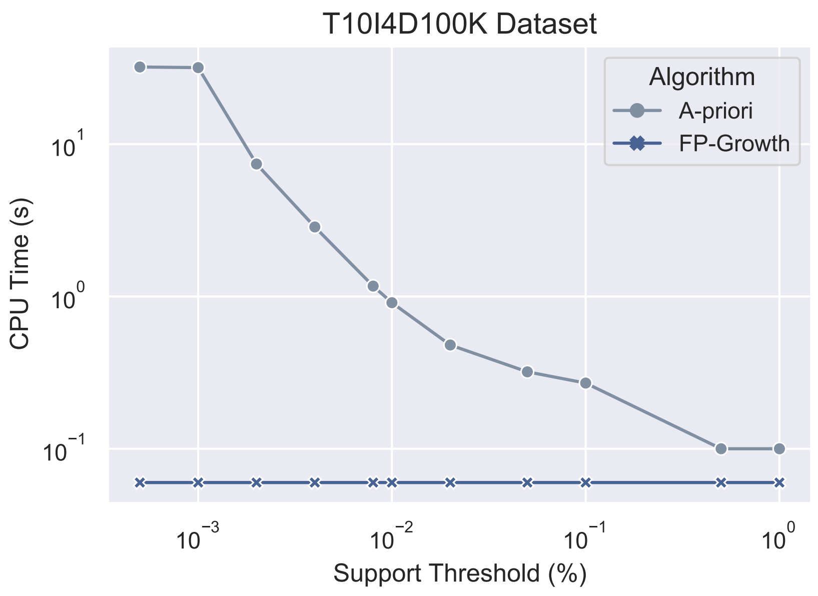

Figure 3.2: A-priori and FP-Growth runtime comparison for the T10I4D100K dataset.

3.3 The retail dataset

As a final experiment for the purposes of this comparative analysis between the two algorithms' runtimes, the same procedure is followed for a third dataset, namely retail. This dataset is a donation by Tom Brijs to the 2003 ICDM Workshop of Frequent Itemset Mining Implementations and contains the anonymized retail market basket data from an anonymous Belgian retail store. The number of unique items is 16470 and the total number of transactions is 88162. Since the number of transactions is similar to that of the T10I4D100K dataset, but with a considerably higher number of unique items, the execution of both algorithms is expected to require more time.

t (%)	0.0005	0.001	0.002	0.004	0.008	0.01	0.02	0.05	0.1	0.5	1
A-priori (s)	68.23	67.92	8.25	2.27	0.96	0.83	0.5	0.27	0.13	0.09	0.09
FP-Growth (s)	0.09	0.09	0.09	0.09	0.09	0.09	0.08	0.08	0.08	0.08	0.08

Table 3.3: Runtime results for the retail dataset.

Indeed, the total time required for the convergence of both algorithms seems to have increased, however the results - which can be seen in Table 3.3 and Fig. 3.3 - are consistent with the ones acquired for the T10I4D100K dataset: FP-Growth converges at a threshold-independent time of $\mathcal{O}(0.1)$ s, while a-priori's runtime varies with the support threshold's value.

The fact that all times are scaled with a factor of ~ 2, while the number of transactions is relatively smaller compared to the number of transactions in T10I4D100K is because the number of unique items and the total number of transactions are more of an indication rather than a measure themselves regarding the algorithm's runtime. While these values are used in order to determine runtime overestimations in terms of the algorithms' time complexities, what matters at the end of the day is the number of frequent itemsets, which can be very low even at large datasets or very high even at small datasets. To illuminate this point, two additional experiments are carried out, both of which are presented in the final section of this investigation.

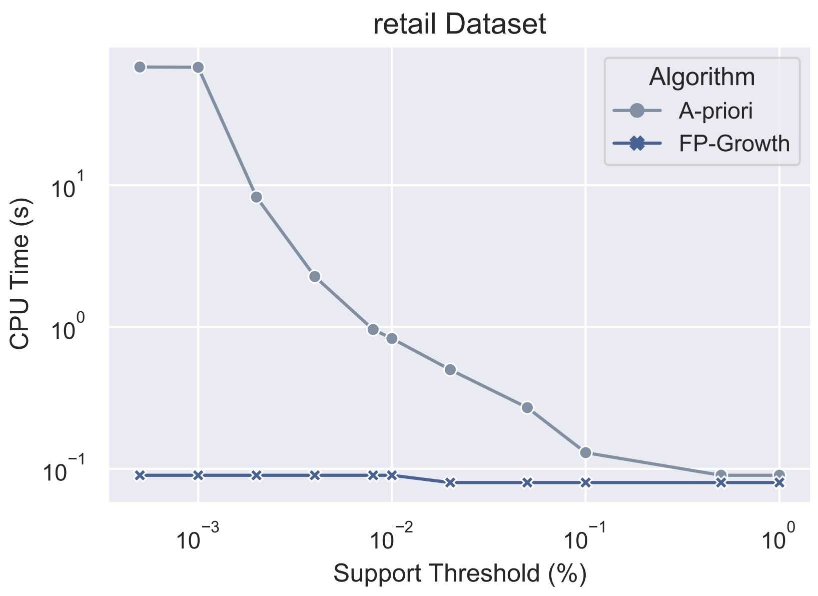

Figure 3.3: A-priori and FP-Growth runtime comparison for the retail dataset.

3.4 The chess and connect datasets

The datasets chosen for the two final experiments are chess and connect. Both of them were also prepared by Roberto Bayardo from the UCI datasets and PUMSB, with chess containing only 75 unique items and 3196 transactions and connect containing 129 unique items and 67557 transactions. If the number of unique items or the number of transactions were enough to determine the algorithms' runtime, one would expect the a-priori algorithm to converge in less than 30 seconds with a support threshold of 0.0005% for both datasets. However, an experiment performed for each of these datasets using the a-priori algorithm with a support threshold of 0.001% (two times the value of the minimum support threshold in the previous experiments) showed that convergence was not achieved even after 6 hours of CPU time, at which point the algorithm was still evaluating frequent 11-itemsets. As explained above, the reason for this is that the number of frequent itemsets is much larger in these datasets compared to the ones previously studied: due to their nature[4], these datasets are expected to consist of many transactions that share a lot of common items, which is why the number of frequent itemsets turns out to be rather large. On the other hand, as far as FP-Growth's performance is concerned, the results are shown in Table 3.4 and depicted in logarithmic scales in Fig. 3.4.

t (%)	0.0005	0.001	0.002	0.004	0.008	0.01	0.02	0.05	0.1	0.5	1
chess (s)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
connect (s)	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13

Table 3.4: FP-Growth's runtime results for the chess and connect datasets.

Once again, FP-Growth deals with the problem of frequent itemset mining in just a fraction of a second for each dataset, which is again independent of the support threshold's value. It is worth noting at this point that writing the frequent itemsets on the disk requires more than 100 GB of memory and

[4] For example, while details are not available, the chess dataset is probably related to the sequence of moves in a chess game. Many games follow very similar sequences during the first 10-20 moves, so each transaction of the chess dataset may be identical to many others as far as the first 10-20 items are concerned.

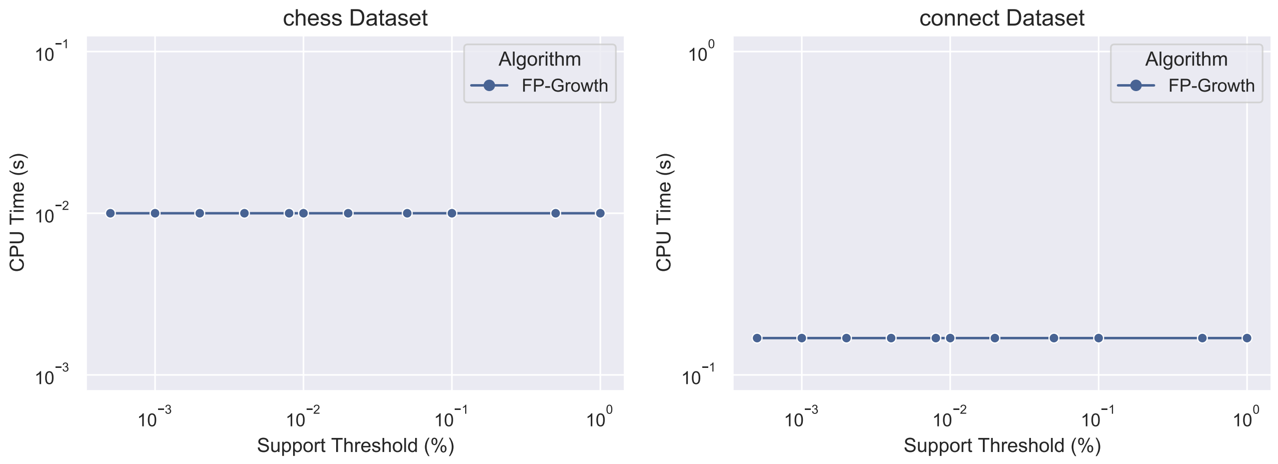

Figure 3.4: The runtimes of FP-Growth for the chess and connect datasets.

around 5 minutes of time, since their number is indeed very large (higher than 10^{17}), despite the small volume of the two databases.

Closing this analysis, it is apparent that, while both algorithms studied are highly optimized versions of the original algorithms shown in class, FP-Growth is undoubtedly more efficient, being able to determine even quadrillions of frequent itemsets in record times.

4 Link Analysis Exercises

4.1 Exercise 5.2.2, MMDS

The two graphs considered are the ones shown in Fig. 4.1, where the graph of Fig. 5.4 of [4] is shown on the left and the graph of Fig. 5.7 of [4] is shown on the right.

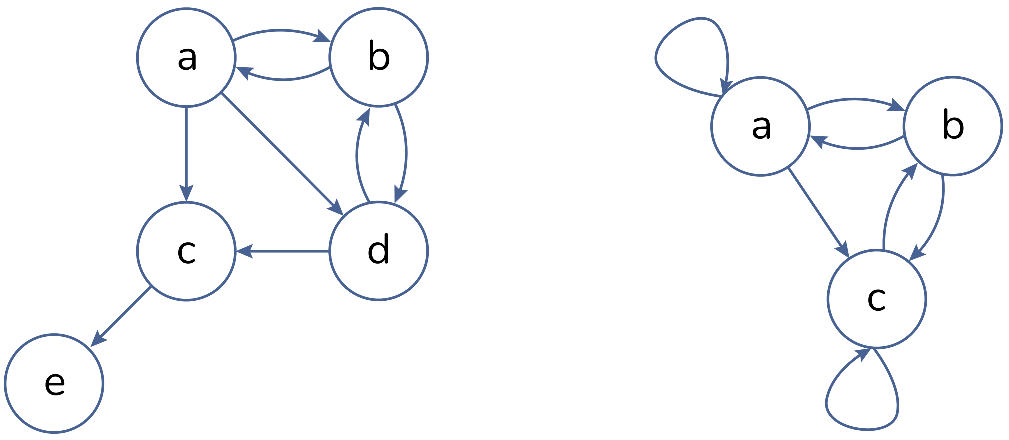

Figure 4.1: The graph of Fig. 5.4 of [4] (left) and the graph of Fig. 5.7 of [4] (right).

Given the states $\{a, b, c, d, e\}$ and $\{a, b, c\}$ for the first and second graphs, respectively, the corresponding transition matrices, as represented by the out-degree of each node and the list of its successors, are shown in Table 4.1.

Source	Degree	Destinations
a	3	b, c, d
b	2	a, d
c	1	e
d	2	b, c
e	0	$\emptyset$

Source	Degree	Destinations
a	3	a, b, c
b	2	a, c
c	2	b, c

Table 4.1: The transition matrices for the graphs shown in Fig. 4.1.

4.2 Exercise 5.2.3, MMDS

The graph considered in this exercise is the one shown in Fig. 4.2.

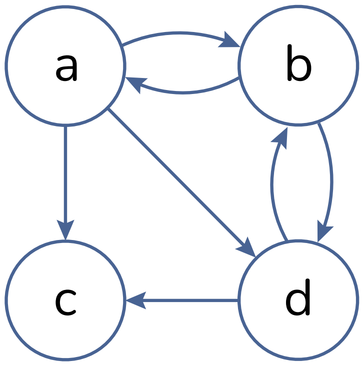

Figure 4.2: The graph of Fig. 5.3 of [4].

This four-node graph is divided into four 2×2 square blocks. The first, block M_{11}, consists of the part of the graph's transition matrix with starting pages a, b and destinations a, b, as shown in Table 4.2.

Source	Degree	Destinations
a	3	b
b	2	a

Table 4.2: The M_{11} block of the transition matrix of the graph shown in Fig. 4.2.

The second block, M_{12}, consists of the part of the graph's transition matrix with starting pages c, d and destinations a, b, as shown in Table 4.3.

Source	Degree	Destinations
d	2	b

Table 4.3: The M_{12} block of the transition matrix of the graph shown in Fig. 4.2.

The reason why there is no entry with c as a source is because there are no out-links from the c state in the graph of Fig. 4.2. As far as the third block, M_{21}, is concerned, it consists of the part of the graph's transition matrix with starting pages a, b and destinations c, d and is shown in Table 4.4.

Source	Degree	Destinations
a	3	c, d
b	2	d

Table 4.4: The M_{21} block of the transition matrix of the graph shown in Fig. 4.2.

Finally, the M_{22} block consists of the part of the graph's transition matrix with starting pages c, d and destinations c, d and is shown in Table 4.5. Again, there is no c-source because the graph of Fig. 4.2 has no out-links from the c-state.

Source	Degree	Destinations
d	2	c

Table 4.5: The M_{22} block of the transition matrix of the graph shown in Fig. 4.2.

4.3 Exercise 5.3.1, MMDS

The studied graph for the purposes of this exercise is the one shown in Fig. 4.3.

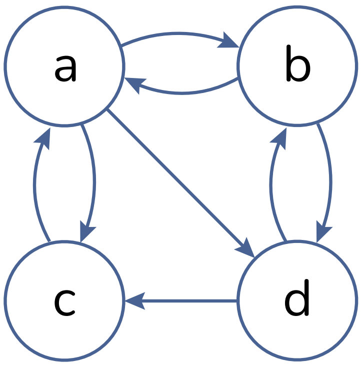

Figure 4.3: The graph of Fig. 5.15 of [4].

The transition matrix, $\mathbf{M}$, corresponding to this graph is

$$\mathbf{M} = \begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}. \tag{4.3.1}$$

Given the transition matrix, the topic-sensitive Page-Rank, $\mathbf{v}$, is the solution of

$$\mathbf{v} = \beta \mathbf{M} \cdot \mathbf{v} + \frac{1-\beta}{|S|} \mathbf{e}_S, \tag{4.3.2}$$

where $0 < \beta < 1$, S is the teleport set and $\mathbf{e}_S$ is the binary vector that has 1 in the components in S and 0 in other components. Supposing that the matrix $\mathbb{1} - \beta \mathbf{M}$ is invertible, the topic-sensitive Page-Rank can be written as

$$\mathbf{v} = \frac{1-\beta}{|S|}\left(\mathbb{1} - \beta\mathbf{M}\right)^{-1} \cdot \mathbf{e}_S. \tag{4.3.3}$$

(a) For the first case, the teleport set is $S = \{a\}$, therefore $|S| = 1$ and $\mathbf{e}_S = [1, 0, 0, 0]^\top$. Supposing that $\beta = 0.8$, as in Example 5.10 of [4], Eq. (4.3.3) becomes

$$\mathbf{v} = \frac{1}{5}\begin{pmatrix} 1 & -2/5 & -4/5 & 0 \\ -4/15 & 1 & 0 & -2/5 \\ -4/15 & 0 & 1 & -2/5 \\ -4/15 & -2/5 & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.3.4}$$

and since

$$\begin{pmatrix} 1 & -2/5 & -4/5 & 0 \\ -4/15 & 1 & 0 & -2/5 \\ -4/15 & 0 & 1 & -2/5 \\ -4/15 & -2/5 & 0 & 1 \end{pmatrix}^{-1} = \frac{1}{7}\begin{pmatrix} 15 & 66/7 & 12 & 60/7 \\ 20/3 & 263/21 & 16/3 & 50/7 \\ 20/3 & 116/21 & 37/3 & 50/7 \\ 20/3 & 158/21 & 16/3 & 85/7 \end{pmatrix}, \tag{4.3.5}$$

the topic-sensitive Page-Rank is

$$\mathbf{v} = \left[\frac{3}{7}, \frac{4}{21}, \frac{4}{21}, \frac{4}{21}\right]^\top. \tag{4.3.6}$$

This result can also be acquired via the iterative procedure followed in class. Supposing that one starts with a random vector, for example $\mathbf{v}^{(0)} = [1/4, 1/4, 1/4, 1/4]^\top$, Eq. (4.3.2) can be written as an iterative equation as follows:

$$\mathbf{v}^{(k+1)} = \beta\mathbf{M} \cdot \mathbf{v}^{(k)} + \frac{1-\beta}{|S|}\mathbf{e}_S, \quad k \geq 0. \tag{4.3.7}$$

The topic-sensitive Page-Rank can be acquired by iteratively calculating $\mathbf{v}$ for $k = 1, 2, \ldots, k_F$, where k_F is the iteration for which

$$|v_i^{(k_F)} - v_i^{(k_F-1)}| < \epsilon, \quad \forall\, i = 1, \ldots, 4 \tag{4.3.8}$$

holds for the first time. In Eq. (4.3.8), ϵ is a small number which corresponds to the convergence criterion. Choosing $\epsilon = 10^{-3}$, the results of each iteration starting from $\mathbf{v}^{(0)}$ are shown below:

$$\underbrace{\begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix}}_{k=0}, \underbrace{\begin{pmatrix} 0.5 \\ 0.167 \\ 0.167 \\ 0.167 \end{pmatrix}}_{k=1}, \underbrace{\begin{pmatrix} 0.4 \\ 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}}_{k=2}, \underbrace{\begin{pmatrix} 0.44 \\ 0.187 \\ 0.187 \\ 0.187 \end{pmatrix}}_{k=3}, \underbrace{\begin{pmatrix} 0.424 \\ 0.192 \\ 0.192 \\ 0.192 \end{pmatrix}}_{k=4}, \underbrace{\begin{pmatrix} 0.43 \\ 0.19 \\ 0.19 \\ 0.19 \end{pmatrix}}_{k=5}, \underbrace{\begin{pmatrix} 0.428 \\ 0.191 \\ 0.191 \\ 0.191 \end{pmatrix}}_{k=6}, \underbrace{\begin{pmatrix} 0.429 \\ 0.19 \\ 0.19 \\ 0.19 \end{pmatrix}}_{k=7}, \underbrace{\begin{pmatrix} 0.428 \\ 0.191 \\ 0.191 \\ 0.191 \end{pmatrix}}_{k=8}$$

Evidently, this iterative process converges after 8 steps to the vector $\mathbf{v} = [0.428, 0.191, 0.191, 0.191]^\top$, which is the 3rd-decimal approximation of the vector of Eq. (4.3.6). Both approaches - the exact solution of the system and the iterative approximate solution - can be generalized for other values of β with the Python code presented in Snippet 1.

```python
import numpy as np
from numpy.linalg import inv

def TSPRCalc(beta=0.8,method='exact',e=[1,0,0,0],thresh=1e-3):
    """
    Args:
        beta (float):
            float between 0 and 1 introduced for teleportation
        method (string):
            either 'exact', to solve the system by matrix inversion
            or 'approx', to calculate the solution iteratively
        e (list):
            list of 0,1 elements corresponding to the teleport set
    """
    M = np.array([[0.0, 0.5, 1.0, 0.0],
                  [1/3, 0.0, 0.0, 0.5],
                  [1/3, 0.0, 0.0, 0.5],
                  [1/3, 0.5, 0.0, 0.0]])
    N = ((1.0-beta)/sum(e))*np.array(e)
    if method=='exact':
        invmat = inv(np.eye(4)-beta*M)
        v = np.matmul(invmat,N)
    else:
        steps = 0
        epsilon = 1.0
        v = np.array([0.25, 0.25, 0.25, 0.25])
        while epsilon > thresh:
            v_new = np.matmul(beta*M,v) + N
            epsilon = abs(v_new-v).max()
            steps += 1
            v = v_new

        print(f'The algorithm converged after {steps} steps.')

    return v
```

Python Code Snippet 1: Topic-sensitive Page-Rank computation

(b) For the second case, the teleport set is $S = \{a, c\}$, therefore $|S| = 2$ and $\mathbf{e}_S = [1, 0, 1, 0]^\top$. Supposing again that $\beta = 0.8$, the inverse matrix of Eq. (4.3.3) has already been calculated in Eq. (4.3.5). As a result, Eq. (4.3.3) now reads

$$\mathbf{v} = \frac{1}{10} \cdot \frac{1}{7} \begin{pmatrix} 15 & {}^{66}/_{7} & 12 & {}^{60}/_{7} \\ {}^{20}/_{3} & {}^{263}/_{21} & {}^{16}/_{3} & {}^{50}/_{7} \\ {}^{20}/_{3} & {}^{116}/_{21} & {}^{37}/_{3} & {}^{50}/_{7} \\ {}^{20}/_{3} & {}^{158}/_{21} & {}^{16}/_{3} & {}^{85}/_{7} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}. \tag{4.3.9}$$

Consequently, the topic-sensitive Page-Rank is

$$\mathbf{v} = \left[\frac{27}{70}, \frac{6}{35}, \frac{19}{70}, \frac{6}{35}\right]^\top. \tag{4.3.10}$$

As far as the iterative method is concerned, given $\epsilon = 10^{-3}$, the results of each iteration starting from $\mathbf{v}^{(0)}$ are shown below:

$$\underbrace{\begin{pmatrix}0.25\\0.25\\0.25\\0.25\end{pmatrix}}_{k=0},\ \underbrace{\begin{pmatrix}0.4\\0.167\\0.267\\0.167\end{pmatrix}}_{k=1},\ \underbrace{\begin{pmatrix}0.38\\0.173\\0.273\\0.173\end{pmatrix}}_{k=2},\ \underbrace{\begin{pmatrix}0.388\\0.171\\0.271\\0.171\end{pmatrix}}_{k=3},\ \underbrace{\begin{pmatrix}0.385\\0.172\\0.272\\0.172\end{pmatrix}}_{k=4},\ \underbrace{\begin{pmatrix}0.386\\0.171\\0.271\\0.171\end{pmatrix}}_{k=5},\ \underbrace{\begin{pmatrix}0.386\\0.171\\0.271\\0.171\end{pmatrix}}_{k=6}$$

In this case, convergence is achieved after 6 steps, resulting in the vector $\mathbf{v} = [0.386, 0.171, 0.271, 0.171]^\top$, which is the 3rd-decimal approximation of the vector of Eq. (4.3.10).

4.4 Exercise 5.4.2, MMDS

The studied graph for the purposes of this exercise is the same as the one portrayed in Fig. 4.3 of the previous exercise, so the corresponding transition matrix is that of Eq. (4.3.1).

(a) The Trust-Rank is nothing but the topic-sensitive Page-Rank, where the teleport set consists of the trusted pages. In the case where only b is a trusted page, the Trust-Rank vector, $\mathbf{v}$, is given by

$$\mathbf{v} = \frac{1}{5}\cdot\frac{1}{7}\begin{pmatrix}15 & ^{66}/_{7} & 12 & ^{60}/_{7}\\ ^{20}/_{3} & ^{263}/_{21} & ^{16}/_{3} & ^{50}/_{7}\\ ^{20}/_{3} & ^{116}/_{21} & ^{37}/_{3} & ^{50}/_{7}\\ ^{20}/_{3} & ^{158}/_{21} & ^{16}/_{3} & ^{85}/_{7}\end{pmatrix}\cdot\begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \tag{4.4.1}$$

where Eq. (4.3.5) has been substituted in Eq. (4.3.3) and $\beta = 0.8$ has been assumed. The result is

$$\mathbf{v} = \left[\frac{66}{245}, \frac{263}{735}, \frac{116}{735}, \frac{158}{735}\right]^\top. \tag{4.4.2}$$

As in the previous exercise, this result can be obtained by the iterative method described by Eqs. (4.3.7), (4.3.8). With a threshold of $\epsilon = 10^{-3}$, the results of each iteration starting from $\mathbf{v}^{(0)} = [^1/_4, ^1/_4, ^1/_4, ^1/_4]^\top$ are shown below:

$$\underbrace{\begin{pmatrix}0.25\\0.25\\0.25\\0.25\end{pmatrix}}_{k=0},\ \underbrace{\begin{pmatrix}0.3\\0.367\\0.167\\0.167\end{pmatrix}}_{k=1},\ \underbrace{\begin{pmatrix}0.28\\0.347\\0.147\\0.227\end{pmatrix}}_{k=2},\ \underbrace{\begin{pmatrix}0.256\\0.365\\0.165\\0.213\end{pmatrix}}_{k=3},\ \underbrace{\begin{pmatrix}0.278\\0.354\\0.154\\0.214\end{pmatrix}}_{k=4},\ \underbrace{\begin{pmatrix}0.264\\0.36\\0.16\\0.216\end{pmatrix}}_{k=5},\ \underbrace{\begin{pmatrix}0.272\\0.357\\0.157\\0.214\end{pmatrix}}_{k=6},\ \underbrace{\begin{pmatrix}0.268\\0.358\\0.158\\0.215\end{pmatrix}}_{k=7},\ \underbrace{\begin{pmatrix}0.27\\0.358\\0.158\\0.215\end{pmatrix}}_{k=8},\ \underbrace{\begin{pmatrix}0.269\\0.358\\0.158\\0.215\end{pmatrix}}_{k=9}$$

Convergence is achieved after 9 steps, resulting in the vector $\mathbf{v} = [0.269, 0.358, 0.158, 0.215]^\top$, which is the 3rd-decimal approximation of the vector of Eq. (4.4.2).

(b) In order to determine each page's spam mass, the corresponding (ordinary) Page-Ranks are also required. These are acquired by setting $\beta = 1$ in Eq. (4.3.2), thus transforming it into an eigenvectors-eigenvalues problem, where the eigenvalue is equal to one and the eigenvector is normalized to unity:

$$\mathbf{M}\cdot\mathbf{u} = \mathbf{u}, \qquad \|\mathbf{u}\|_2 = 1 \tag{4.4.3}$$

The solution of Eqs. (4.4.3) is

$$\mathbf{u} = \left[\frac{1}{3}, \frac{2}{9}, \frac{2}{9}, \frac{2}{9}\right]^\top \tag{4.4.4}$$

and can be obtained either by direct calculation, or by an iterative method similar to the one described by Eqs. (4.3.7), (4.3.8), where $\beta = 1$. Indeed, with a threshold of 10^{-3}, the results of each iteration starting from $\mathbf{u}^{(0)} = [1/4, 1/4, 1/4, 1/4]^\top$ are:

$$\underbrace{\begin{pmatrix}0.25\\0.25\\0.25\\0.25\end{pmatrix}}_{k=0}, \underbrace{\begin{pmatrix}0.375\\0.208\\0.208\\0.208\end{pmatrix}}_{k=1}, \underbrace{\begin{pmatrix}0.312\\0.229\\0.229\\0.229\end{pmatrix}}_{k=2}, \underbrace{\begin{pmatrix}0.344\\0.219\\0.219\\0.219\end{pmatrix}}_{k=3}, \underbrace{\begin{pmatrix}0.328\\0.224\\0.224\\0.224\end{pmatrix}}_{k=4}, \underbrace{\begin{pmatrix}0.336\\0.221\\0.221\\0.221\end{pmatrix}}_{k=5}, \underbrace{\begin{pmatrix}0.332\\0.223\\0.223\\0.223\end{pmatrix}}_{k=6}, \underbrace{\begin{pmatrix}0.334\\0.222\\0.222\\0.222\end{pmatrix}}_{k=7}, \underbrace{\begin{pmatrix}0.333\\0.222\\0.222\\0.222\end{pmatrix}}_{k=8}$$

At this point, given each page's ordinary Page-Rank, r, as well as its Trust-Rank, t, the spam mass can be calculated as $(r - t)/r$. The results for each page's spam mass can be seen in Table 4.6.

Page	Page-Rank	Trust-Rank	Spam Mass
a	$1/3$	$66/245$	0.192
b	$2/9$	$263/735$	-0.61
c	$2/9$	$116/735$	0.29
d	$2/9$	$158/735$	0.033

Table 4.6: Calculation of each page's spam mass.

The fact that only the spam mass of page b is negative is consistent with the fact that page b is the only trusted source.

5 Clustering Exercises

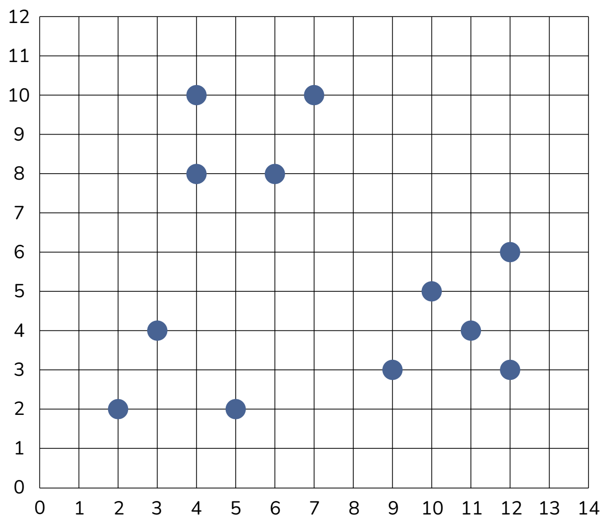

Figure 5.1: The points considered for the studied clustering exercises.

The points considered for the following clustering exercises (Example 7.2 of [4]) are shown on the grid of Fig. 5.1. The three clusters formed can be easily discerned.

5.1 Exercise 7.2.2, MMDS

Before moving on with the clustering of the points of Fig. 5.1 with the two proposed metrics, it is noted here that there are more than one equivalent sequences of clustering steps. For example, when clustering points via the minimum distance criterion, the minimum distance may be achieved by more than one pairs of points. Such "ties" are broken randomly, therefore only one of these equivalent sequences is presented below for each question.

(a) Starting by assigning a cluster to each point, the first merging that occurs based on the minimum distance between any two points - one from each cluster - is that of points $(11, 4)$ and $(12, 3)$, the distance of which is $\sqrt{2}$. Then, the point $(10, 5)$ is also added to this cluster, since the minimum distance is also $\sqrt{2}$. The next merge occurs between $(4, 8)$ and $(6, 8)$, the distance of which is 2. Consequently, $(4, 10)$ is also added to this cluster, with a minimum distance equal to 2 as well (so far, two of the aforementioned ties have already come up). The next two steps consist of the addition of $(12, 6)$ and $(9, 3)$ to the cluster of $\{(10, 5), (11, 4), (12, 3)\}$, both of which have a minimum distance of $\sqrt{5}$. The point $(7, 10)$ is added to the cluster formed by $\{(4, 8), (6, 8), (4, 10)\}$, since the minimum distance is also $\sqrt{5}$. The distance between $(2, 2)$ and $(3, 4)$ is also $\sqrt{5}$, so these two points form their own cluster as well. With a slightly higher minimum distance, equal to $\sqrt{8}$, the point $(5, 2)$ is then added to the cluster $\{(2, 2), (3, 4)\}$. At this point, the three clusters have been correctly identified. However, if one proceeds to merge the clusters together[5], then the order in which they are merged does not matter, as the minimum distance is equal to $\sqrt{17}$ in both cases. This hierarchical clustering process is summarized in the graph of Fig. 5.2, where the sequence is depicted as merges of increasing height, from the bottom to the top.

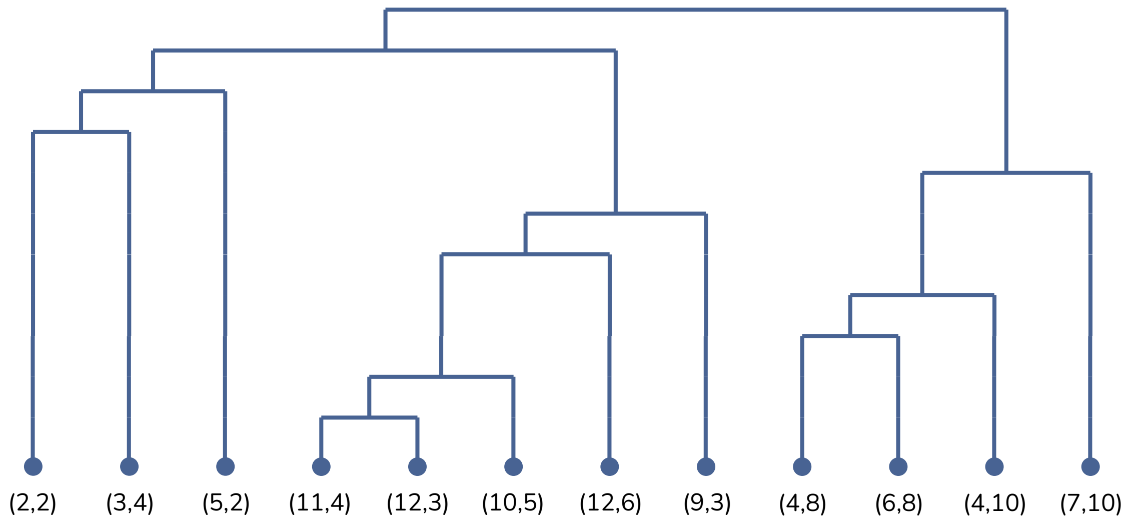

Figure 5.2: Clustering based on the minimum pairwise inter-cluster distance.

(b) When the criterion is switched to the average pairwise inter-cluster distance, the first step is the merging of $(11, 4)$ and $(10, 5)$ into a single cluster, with an (average) distance of $\sqrt{2}$. The next merge is between $(6, 8)$ and $(4, 8)$, the (average) distance of which is equal to 2. Then, $(12, 3)$ is added to the cluster $\{(11, 4), (10, 5)\}$, having an average distance of $\sqrt{4.5}$. The (average) distance between $(2, 2)$ and $(3, 4)$ is $\sqrt{5}$, so these two points form their own cluster. Consequently, $(4, 10)$ is added to the cluster $\{(6, 8), (4, 8)\}$, with an average distance of $1 + \sqrt{2}$. The point $(9, 3)$ is then added in the cluster $\{(11, 4), (10, 5), (12, 3)\}$, with an average distance equal to $1 + 2\sqrt{5}/3$. The next addition is that of $(5, 2)$ into the cluster $\{(2, 2), (3, 4)\}$, with an average distance of $1.5 + \sqrt{2}$. Then, with an average

[5] Note that in most realistic cases the exact number of clusters is not known beforehand.

distance of $\left(3+3\sqrt{2}+2\sqrt{5}\right)/3$, the point $(12,6)$ is added to the cluster $\{(9,3),(11,4),(10,5),(12,3)\}$. With the addition of $(7,10)$ into the cluster $\{(4,10),(6,8),(4,8)\}$, corresponding to an average distance of $\left(3+\sqrt{5}+\sqrt{13}\right)/3$, the three clusters have been formed. If one continues this process, using the average pairwise inter-cluster distance as a clustering criterion, the order in which the three clusters are merged is now unique: first, the clusters $\{(5,2),(2,2),(3,4)\}$ and $\{(7,10),(4,10),(6,8),(4,8)\}$ are merged together, with an average distance approximately equal to 6.842 and then all points are merged into a single cluster. This process is shown in the graph of Fig. 5.3. Note that the order in which the points of Fig. 5.1 are given is not identical to the one shown in the graph of Fig. 5.2.

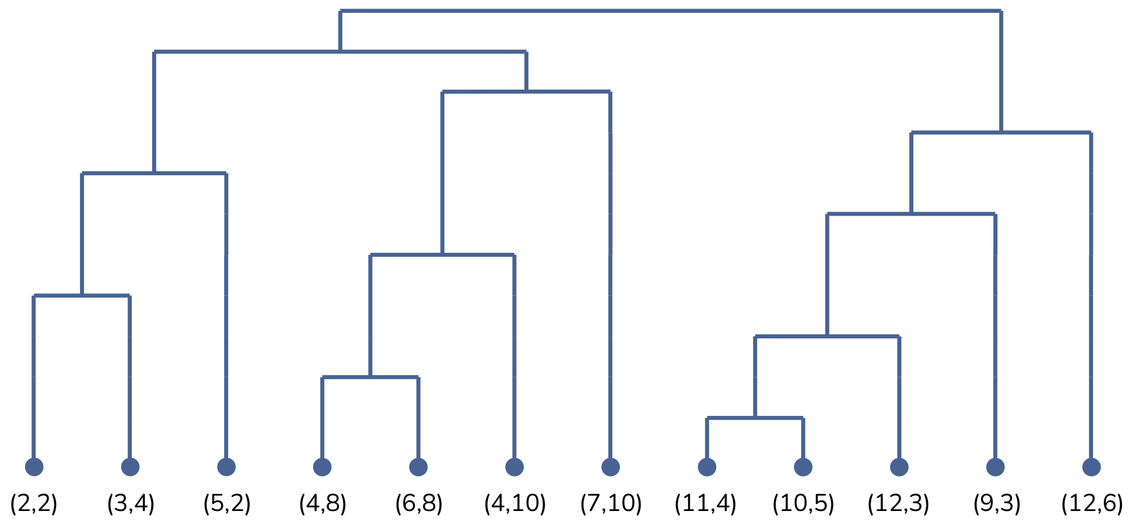

Figure 5.3: Clustering based on the average pairwise inter-cluster distance.

Before closing the present exercise, it is stressed once more that the two sequences presented above are not unique, since many ties were randomly broken. The code shown in Snippet 2 and developed in Python can be used to find other such sequences.

```python
import numpy as np

def dist(a,b):
    a, b = np.array(a), np.array(b)
    return np.sqrt(np.dot(a-b,a-b))

def printdict(diction):
    for key in diction:
        if diction[key] != '':
            print(f'{key} -> {diction[key]}.')

pts = [(2,2), (3,4), (4,8), (4,10), (5,2), (6,8),
       (7,10), (9,3), (10,5), (11,4), (12,3), (12,6)]

def run_clust(pts=[], method='min'):

    ptsdict = dict()
    for idx,elem in enumerate(pts):
        ptsdict[elem] = f'Cluster {idx+1}'

    clustdict = dict()
    for i in range(len(pts)):
        clustdict[f'Cluster {i+1}'] = [pts[i]]

    print('Initial Clustering:')
    printdict(clustdict)
    print('\n')

    clustset = set(clustdict.keys())
    while len(clustset) > 1:
        # Calculate minimums
        run_min = 1e10
        merges = dict()

        if method=='avg':
            for clust_a in clustset:
                for clust_b in clustset:
                    if clust_a != clust_b:
                        pts1, pts2 = clustdict[clust_a], clustdict[clust_b]
                        cter = 0
                        avg_dist = 0.0
                        for pt1 in pts1:
                            for pt2 in pts2:
                                avg_dist += dist(pt1,pt2)
                                cter += 1
                        avg_dist = avg_dist/cter
                        if avg_dist <= run_min:
                            run_min = avg_dist
                            merges[avg_dist] = [clust_a,clust_b]
        else:
            for i in range(len(pts)):
                for j in range(i,len(pts)):
                    point_1, point_2 = pts[i], pts[j]
                    curr_dist = dist(point_1,point_2)
                    if (curr_dist > 1e-5) and (curr_dist <= run_min) and \
                    (ptsdict[point_1] != ptsdict[point_2]):
                        run_min = curr_dist
                        merges[curr_dist] = [ptsdict[point_1],ptsdict[point_2]]

        # perform the merges
        clust_1, clust_2 = merges[run_min] # <- The clusters to merge
        tds = 'Average' if method=='avg' else 'Minimum'
        print(f"Merging {clust_1} with {clust_2}. {tds} distance: {run_min}.")
        clustdict[clust_2] = clustdict[clust_1]+clustdict[clust_2]
        clustdict[clust_1] = ''
        for elem in ptsdict:
            if ptsdict[elem] == clust_1: ptsdict[elem] = clust_2
        print(f'The clusters are now:')
        printdict(clustdict)
        print('\n')
        clustset.remove(clust_1)
```

Python Code Snippet 2: Clustering via minimum or average pairwise inter-cluster distance

5.2 Exercise 7.2.3, MMDS

This exercise is simply an extension of the previous one, where two additional measures are taken into consideration. The followed procedure and the used notation is completely analogous to the one shown in the previous exercise.

(a) When the criterion is switched to the smallest end-cluster radius, the clustering sequence starts with the merging of $(10, 5)$ and $(11, 4)$, resulting into a cluster of radius equal to $\sqrt{2}/2$. Then, $(4, 10)$ and $(4, 8)$ are merged, forming a cluster of radius 1. Consequently, $(7, 10)$ and $(6, 8)$ form their own cluster with a radius of $\sqrt{5}/2$ and so do $(3, 4)$ and $(2, 2)$. Then, $(12, 6)$ is added to the cluster $\{(10, 5), (11, 4)\}$, thus enlarging its radius and making it equal to $\sqrt{2}$. Afterwards, $(9, 3)$ and $(12, 3)$ form a cluster of radius 1.5. The next step is the addition of $(5, 2)$ into the cluster $\{(3, 4), (2, 2)\}$, the radius of which becomes equal to $\sqrt{29}/3$. Then, the clusters $\{(4, 10), (4, 8)\}$ and $\{(7, 10), (6, 8)\}$ are merged into a single cluster with radius approximately equal to 2.02. Finally, with the merging of $\{(12, 6), (10, 5), (11, 4)\}$ and $\{(9, 3), (12, 3)\}$, resulting into a cluster of radius approximately equal to 2.16, the three clusters have been formed. If the clustering process continues, the clusters $\{(5, 2), (3, 4), (2, 2)\}$ and $\{(4, 10), (4, 8), (7, 10), (6, 8)\}$ are merged into a single cluster of radius approximately equal to 4.93, before being merged with the remaining points into a single cluster. The process is depicted in the graph of Fig. 5.4.

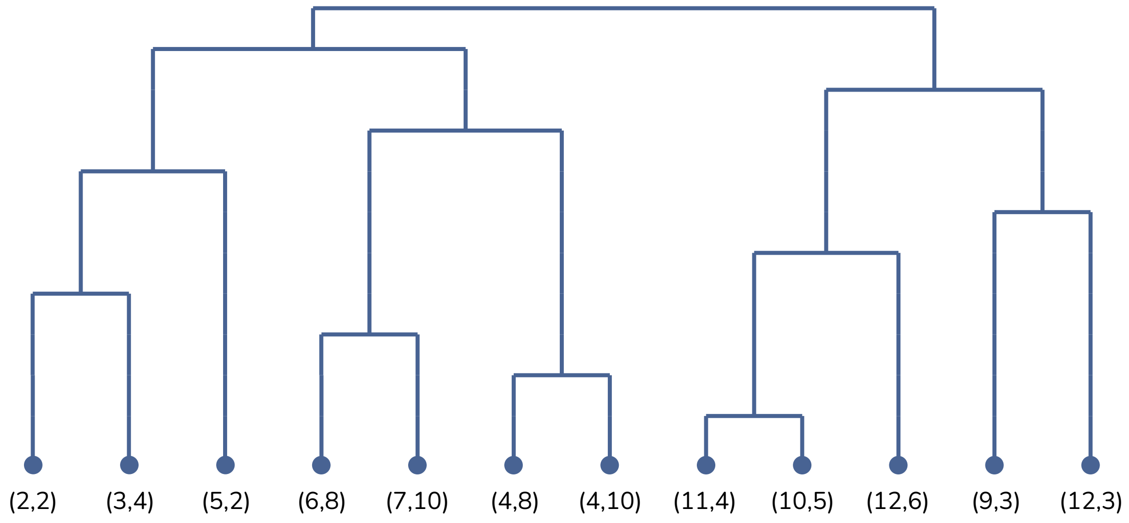

Figure 5.4: Clustering based on minimizing the end-cluster radius.

(b) Finally, using the end-cluster diameter as the criterion, the first step of the clustering sequence consists (again) of the merging of $(10, 5)$ and $(11, 4)$ into a cluster of diameter equal to $\sqrt{2}$. In fact, it's not just the first step, but the whole sequence that is identical to the previous one, even though a cluster's radius is not equal to the half of its diameter and thus the two criteria are not in general equivalent. Points $(4, 10)$ and $(4, 8)$ are merged to form a cluster of diameter 2. Then, $(7, 10)$ and $(6, 8)$ are merged into a cluster of diameter equal to $\sqrt{5}$ and so do points $(3, 4)$ and $(2, 2)$. Once $(12, 6)$ is added to the cluster $\{(10, 5), (11, 4)\}$, its diameter also becomes equal to $\sqrt{5}$. The point $(5, 2)$ is added to the cluster $\{(3, 4), (2, 2)\}$, thus making its diameter equal to 3. Simultaneously, $(9, 3)$ and $(12, 3)$ are merged into a single cluster of diameter equal to 3. The eighth step consists of the merging of $\{(4, 10), (4, 8)\}$ and $\{(7, 10), (6, 8)\}$ into a cluster of diameter $\sqrt{13}$. Then, once $\{(9, 3), (12, 3)\}$ and $\{(12, 6), (10, 5), (11, 4)\}$ are merged into a cluster of diameter $3\sqrt{2}$, the three clusters have been formed. In order to end up with two clusters, $\{(5, 2), (3, 4), (2, 2)\}$ is merged with $\{(4, 10), (4, 8), (7, 10), (6, 8)\}$, forming a cluster with diameter equal to $\sqrt{89}$. A single cluster of diameter equal to $2\sqrt{29}$ can be obtained by merging these two final clusters. Since the clustering sequence is identical to the one presented in the previous question, the corresponding graph is, again, the one shown in Fig. 5.4.

5.3 Exercise 7.3.2, MMDS

Working again with the set of points shown in Fig. 5.1, the first step is the calculation of the diameters and minimum intercluster distances for the clusters shown in Table 5.1.

Cluster	Constituents
$\mathscr{C}_1$	$\{(4,8),(4,10),(6,8),(7,10)\}$
$\mathscr{C}_2$	$\{(2,2),(3,4),(5,2)\}$
$\mathscr{C}_3$	$\{(9,3),(10,5),(11,4),(12,3),(12,6)\}$

Table 5.1: The three clusters formed by the points shown in Fig. 5.1.

As far as $\mathscr{C}_1$ is concerned, its diameter is equal to the distance between $(4,8)$ and $(7,10)$, i.e. $\sqrt{13} \approx 3.61$. Its minimum distance to $\mathscr{C}_2$ is equal to the distance between $(4,8)$ and $(3,4)$, i.e. $\sqrt{17} \approx 4.12$, while its minimum distance to $\mathscr{C}_3$ is equal to the distance between $(6,8)$ and $(10,5)$, i.e. 5. When it comes to the diameter of $\mathscr{C}_2$, it corresponds to the distance between $(3,4)$ and $(5,2)$, i.e. $2\sqrt{2} \approx 2.83$. Its minimum distance to $\mathscr{C}_3$ is equal to the distance between $(5,2)$ and $(9,3)$, i.e. $\sqrt{17} \approx 4.12$. Finally, the diameter of $\mathscr{C}_3$ is equal to the distance between $(12,6)$ and $(9,3)$, i.e. $3\sqrt{2} \approx 4.24$. These results are summarized in Table 5.2.

$\mathscr{C}_i$	$\operatorname{diam}(\mathscr{C}_i)$	$\operatorname{mid}(C_i, C_1)$	$\operatorname{mid}(C_i, C_2)$	$\operatorname{mid}(C_i, C_3)$
$\mathscr{C}_1$	$\sqrt{13}$	0	$\sqrt{17}$	5
$\mathscr{C}_2$	$2\sqrt{2}$	$\sqrt{17}$	0	$\sqrt{17}$
$\mathscr{C}_3$	$3\sqrt{2}$	5	$\sqrt{17}$	0

Table 5.2: The diameters and minimum intercluster distances for the clusters of Table 5.1.

The same quantities are also depicted in Fig. 5.5, where the diameters $\operatorname{diam}(\mathscr{C}_i)$ are drawn with a light blue color and the minimum intercluster distances $\operatorname{mid}(\mathscr{C}_i, \mathscr{C}_j)$ are drawn with a yellow color.

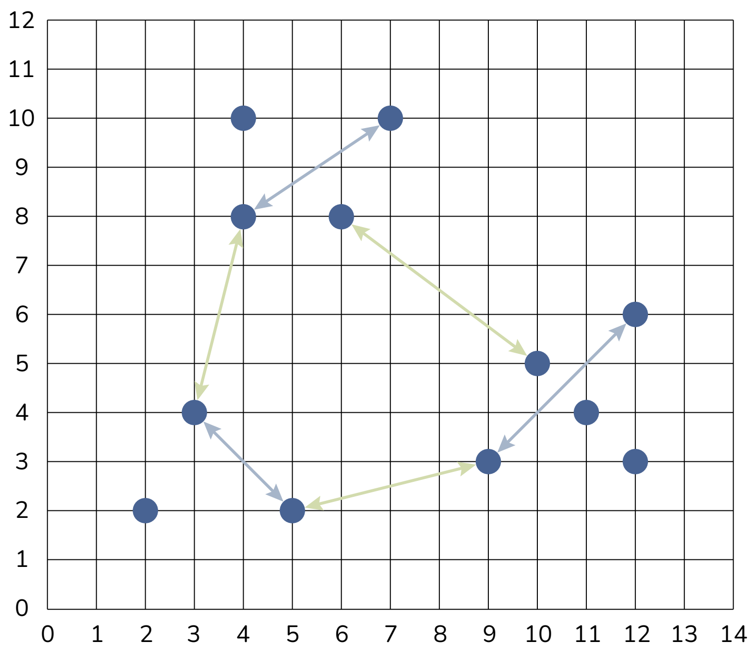

Figure 5.5: The diameters and minimum intercluster distances for the clusters formed by the points.

In general, if the relationship

$$\operatorname{diam}(\mathscr{C}_i) < \operatorname{mid}\left(\mathscr{C}_i, \mathscr{C}_j\right) \tag{5.3.1}$$

holds for all $i \neq j$, then all starting points must belong to different clusters. This relationship implies that if the diameter of each cluster $\mathscr{C}_i$ is smaller than all the minimum intercluster distances involving this cluster, then for any point $\mathbf{p}$ generated in $\mathscr{C}_i$, the point $\mathbf{p}'$ which maximizes $\|\mathbf{p} - \mathbf{p}'\|_2$ cannot also belong in $\mathscr{C}_i$. The proof is trivial; supposing that $\mathbf{p}'$ also belongs in $\mathscr{C}_i$, then

$$\operatorname{mid}\left(\mathscr{C}_i, \mathscr{C}_j\right) < \|\mathbf{p} - \mathbf{p}'\|_2 \tag{5.3.2}$$

must hold for all $j \neq i$, otherwise $\mathbf{p}'$ would be generated in another cluster. However,

$$\|\mathbf{p} - \mathbf{p}'\|_2 \leq \operatorname{diam}(\mathscr{C}_i) \tag{5.3.3}$$

also holds by definition, thus leading to

$$\operatorname{mid}\left(\mathscr{C}_i, \mathscr{C}_j\right) < \operatorname{diam}(\mathscr{C}_i), \tag{5.3.4}$$

which is inconsistent with (5.3.1). As a result, the hypothesis that $\mathbf{p}$ and $\mathbf{p}'$ can be generated in the same cluster while (5.3.1) holds is incorrect.

The problem in the present exercise is that (5.3.1) does not hold, since

$$\operatorname{diam}(\mathscr{C}_3) = 3\sqrt{2} > \sqrt{17} = \operatorname{mid}(\mathscr{C}_2, \mathscr{C}_3). \tag{5.3.5}$$

As a result, the general theorem that was proven above cannot be applied in this case. Instead, a different theorem must be formulated - one which does not involve the minimum intercluster distance, contrary to what is hinted by the exercise. For this purpose, one may define

$$\operatorname{mbid}\left(\mathscr{C}_i, \mathscr{C}_j\right) = \max_{\mathbf{x}_j}\left\{\|\mathbf{x}_i - \mathbf{x}_j\|_2\right\}, \quad \mathbf{x}_i \in \mathscr{C}_i, \quad \mathbf{x}_j \in \mathscr{C}_j, \quad i \neq j, \tag{5.3.6}$$

where $\mathbf{x}_i$ is the point of $\mathscr{C}_i$ which is "closest" to $\mathscr{C}_j$, i.e. one of the two points that determines the minimum intercluster distance between $\mathscr{C}_i$ and $\mathscr{C}_j$. The quantity of Eq. (5.3.6) shall henceforth be referred to as maximum border intercluster distance (mbid)[6]. Note that the distance defined in this way is not symmetric, i.e. $\operatorname{mbid}\left(\mathscr{C}_i, \mathscr{C}_j\right) \neq \operatorname{mbid}\left(\mathscr{C}_j, \mathscr{C}_i\right)$ in general. This point is illustrated in Fig. 5.6, where two clusters $\mathscr{C}_1$ and $\mathscr{C}_2$ are shown in dark and light blue colors, respectively.

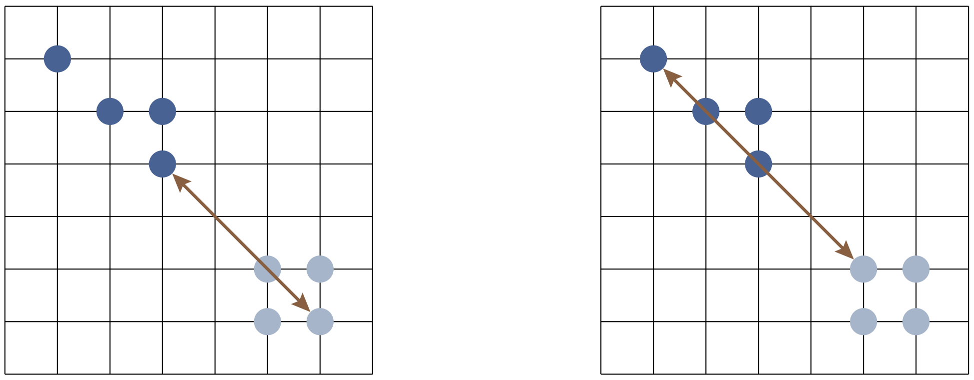

Figure 5.6: An illustration of the non-symmetricity of the maximum border intercluster distance.

[6] Note that this term was coined by the author only for the purposes of the present exercise. To his knowledge, this term does not exist in the relevant bibliography, even though the quantity it represents may exist.

On the left grid of Fig. 5.6, the distance $\mathrm{mbid}(\mathscr{C}_1, \mathscr{C}_2)$ is shown, i.e. the maximum distance between the point of $\mathscr{C}_1$ that is closest to $\mathscr{C}_2$ and any point of $\mathscr{C}_2$. Supposing that the grids are orthonormal, $\mathrm{mbid}(\mathscr{C}_1, \mathscr{C}_2) = 3\sqrt{2}$. On the right grid of Fig. 5.6, the distance $\mathrm{mbid}(\mathscr{C}_2, \mathscr{C}_1)$ is shown, i.e. the maximum distance between the point of $\mathscr{C}_2$ that is closest to $\mathscr{C}_1$ and any point of $\mathscr{C}_1$. In this case, $\mathrm{mbid}(\mathscr{C}_2, \mathscr{C}_1) = 4\sqrt{2} \neq \mathrm{mbid}(\mathscr{C}_1, \mathscr{C}_2)$.

Having defined the maximum border intercluster distance, the theorem that can be formulated and applied in the present exercise is the following: if the relationship

$$\mathrm{diam}(\mathscr{C}_i) < \mathrm{mbid}\left(\mathscr{C}_i, \mathscr{C}_j\right) \tag{5.3.7}$$

holds for all $i \neq j$, then all starting points much belong to different clusters. In order to prove this, suppose that for a generated point $\mathbf{p} \in \mathscr{C}_i$, the point $\mathbf{p}'$ which maximizes $\|\mathbf{p} - \mathbf{p}'\|_2$ also belongs in $\mathscr{C}_i$. The relationship

$$\mathrm{mbid}\left(\mathscr{C}_i, \mathscr{C}_j\right) \leq \max_{\mathbf{x_j}} \left\{\|\mathbf{p} - \mathbf{x}_j\|_2\right\} \tag{5.3.8}$$

holds by definition for all $i \neq j$, with the equality being true only when $\mathbf{p}$ is the point closest to $\mathscr{C}_j$. This relationship simply expresses the trivial fact that any point of $\mathscr{C}_i$ that is not the point closest to $\mathscr{C}_j$ has a larger distance from $\mathscr{C}_j$ than that point. Furthermore, since $\mathbf{p}'$ was generated in $\mathscr{C}_i$,

$$\max_{\mathbf{x_j}} \left\{\|\mathbf{p} - \mathbf{x}_j\|_2\right\} < \|\mathbf{p} - \mathbf{p}'\|_2 \tag{5.3.9}$$

must also hold, otherwise $\mathbf{p}'$ would be generated in another cluster. Finally,

$$\|\mathbf{p} - \mathbf{p}'\|_2 \leq \mathrm{diam}(\mathscr{C}_i) \tag{5.3.10}$$

must also hold by the definition of a cluster's diameter. Combining (5.3.8) - (5.3.10), one arrives at the conclusion

$$\mathrm{mbid}\left(\mathscr{C}_i, \mathscr{C}_j\right) < \mathrm{diam}(\mathscr{C}_i), \tag{5.3.11}$$

which is not in agreement with (5.3.7). Therefore, the initial hypothesis that $\mathbf{p}'$ can be generated in $\mathscr{C}_i$ while (5.3.7) holds is incorrect, thus proving the theorem.

As far as the present exercise is concerned, the maximum border intercluster distances for each pair of clusters are calculated and presented in Table 5.3.

$\mathscr{C}_i$	$\mathrm{diam}(\mathscr{C}_i)$	$\mathrm{mbid}(C_i, C_1)$	$\mathrm{mbid}(C_i, C_2)$	$\mathrm{mbid}(C_i, C_3)$
$\mathscr{C}_1$	$\sqrt{13}$	0	$2\sqrt{10}$	$\sqrt{61}$
$\mathscr{C}_2$	$2\sqrt{2}$	$2\sqrt{13}$	0	$\sqrt{65}$
$\mathscr{C}_3$	$3\sqrt{2}$	$\sqrt{61}$	$5\sqrt{2}$	0

Table 5.3: The diameters and maximum border intercluster distances for the clusters of Table 5.1.

Note that, unlike in Table 5.2 where the sub-table containing the minimum intercluster distances was symmetric, the same cannot be told for the sub-table containing the maximum border intercluster distances of Table 5.3 - a result which is to be expected. Evidently, (5.3.7) holds for all $i \neq j$, therefore the theorem proved above can be applied in the present case. As a result, for the points shown in the grid of Fig. 5.1, any initialization will indeed result in three points each belonging to a different cluster.

5.4 Exercise 7.4.1, MMDS

Departing from the clusters formed by the points shown in Fig. 5.1, the final exercise deals with two clusters, $\mathscr{C}_1$ and $\mathscr{C}_2$, which can be seen in Fig. 5.7. The first cluster, $\mathscr{C}_1$, corresponds to a circle of radius c centered around $(0, 0)$, which is also its centroid. The second cluster, $\mathscr{C}_2$, is a ring of inner radius i and outer radius o ($o > i$), which surrounds the first cluster ($i > c$) and is also centered around $(0, 0)$. As a result, the centroid of $\mathscr{C}_2$ is also $(0, 0)$.

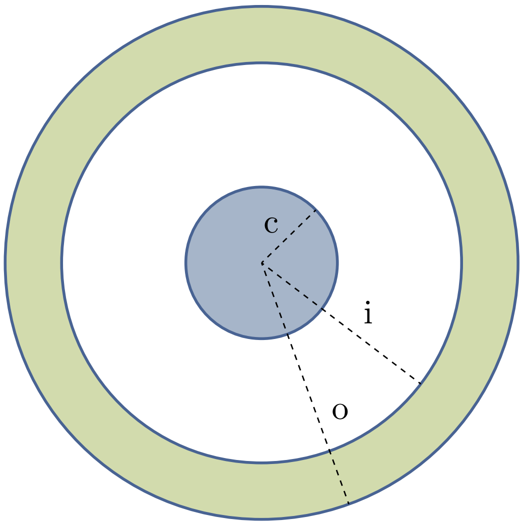

Figure 5.7: A circular cluster of points, $\mathscr{C}_1$, with radius c (blue color) and a surrounding ring, $\mathscr{C}_2$, with inner and outer radius equal to i and o, respectively (yellow color).

In the context of the CURE algorithm, the representative points are chosen such that they all lie on the boundaries of the clusters. This means that all representative points of $\mathscr{C}_1$ lie on the circle's circumference, while the representative points of $\mathscr{C}_2$ lie either on the inner or on the outer circumference of the ring. Given two representative points $\mathbf{x}_1 \in \mathscr{C}_1$, $\mathbf{x}_2 \in \mathscr{C}_2$, their original distance is given by

$$\delta\left(\mathbf{x}_1, \mathbf{x}_2\right) = ||\mathbf{x}_1 - \mathbf{x}_2||_2. \tag{5.4.1}$$

Once both points are moved by 20% of their initial position towards $(0, 0)$, their new positions will be

$$\mathbf{x}_i' = 0.8\mathbf{x}_i, \quad i = 1, 2, \tag{5.4.2}$$

irregardless of where they are approaching $(0, 0)$ from (their displacement's direction is radial, therefore the corresponding vectors are simply scaled). As a result, their new distance will be

$$\delta\left(\mathbf{x}_1', \mathbf{x}_2'\right) = ||\mathbf{x}_1' - \mathbf{x}_2'||_2 = ||0.8\mathbf{x}_1 - 0.8\mathbf{x}_2||_2 = 0.8\delta\left(\mathbf{x}_1, \mathbf{x}_2\right). \tag{5.4.3}$$

After all representative points are moved in this manner, if $\delta\left(\mathbf{x}_1', \mathbf{x}_2'\right) \leq d$ holds for any pair of points, then the two clusters will be merged. In other words, the condition for the two clusters to be merged into a single cluster is

$$0.8\xi := \min_{\mathbf{x}_1, \mathbf{x}_2} \left\{\delta\left(\mathbf{x}_1', \mathbf{x}_2'\right)\right\} \leq d, \tag{5.4.4}$$

where ξ can be identified as the minimum **initial** distance between any two representative points that lie on different clusters. What remains is expressing ξ in terms of c, i and o. However, a general relationship between these quantities cannot be given, since ξ depends on the number, N, as well as the choice of representative points per cluster. For example, if only 1 representative point is elected for each cluster

($N = 1$), ξ belongs in the range $[i - c, c + o]$, where the value $c + o$ corresponds to the case shown in the left of Fig. 5.8 and the value $i - c$ corresponds to the case shown in the right of Fig. 5.8.

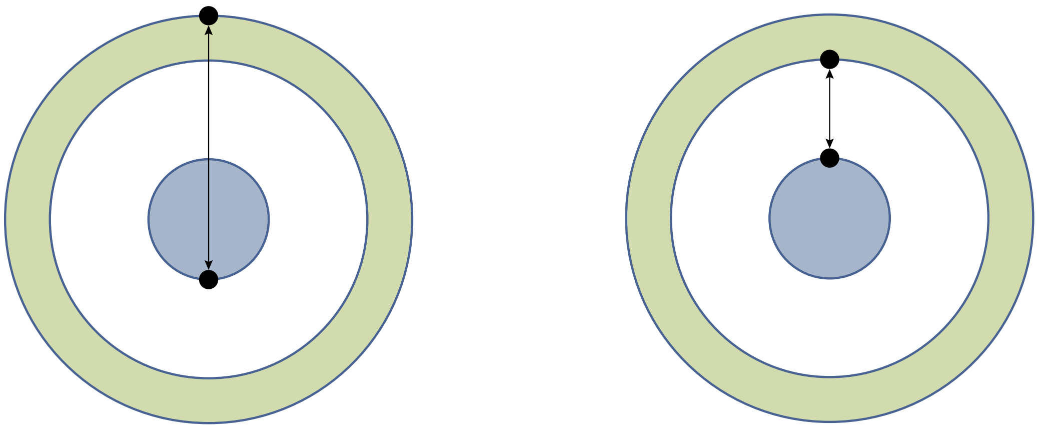
Figure 5.8: The two extreme cases for ξ in the case of 1 representative point per cluster.

In this case, if one ensures that (5.4.4) holds for $\xi = c + o$ (the highest possible value of ξ), then the clusters will be merged into one with absolute certainty, irregardless of the two representative points' initialization. Summarizing, in the case $N = 1$, if

$$d \geq 0.8\,(c + o) \tag{5.4.5}$$

holds, then the two clusters will certainly be merged into one (without this meaning that the merging of the clusters can't be achieved for smaller values of d, for specific choices of the representative points).

As the number of each cluster's representative points increases, the lower bound of ξ remains the same, however the upper bound decreases. This is due to the fact the representative points within each cluster are elected in such a way that the distance between two consecutive points is maximized. For example, in the case of 2 representative points per cluster ($N = 2$), the maximum value of ξ is equal to $\sqrt{c^2 + o^2}$ and can be obtained in the configuration shown in the left of Fig. 5.9.

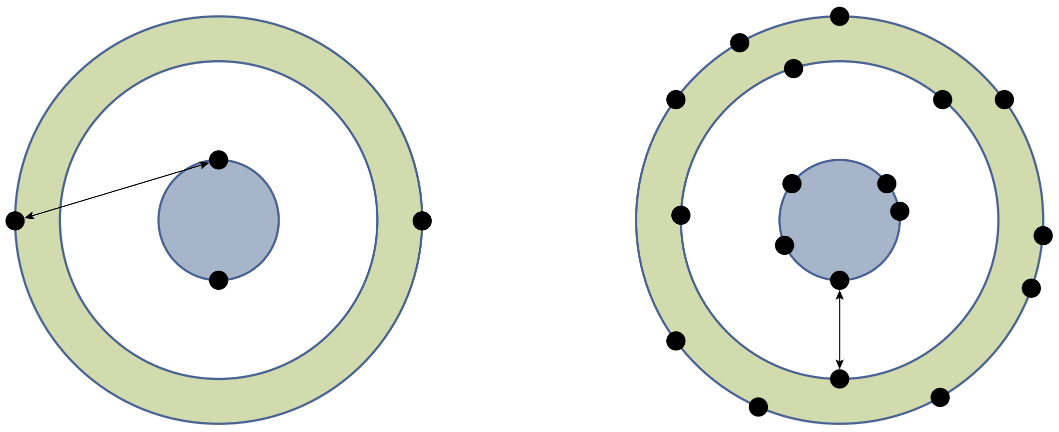
Figure 5.9: Depiction of the upper bound of ξ when 2 representative points are elected per cluster (left) and the lower bound of ξ for larger numbers of representative points (right).

In this case, as long as

$$d \geq 0.8\sqrt{c^2 + o^2} \tag{5.4.6}$$

holds, the two clusters will always be merged. For considerably larger numbers of representative points per cluster, the upper bound of ξ approaches its (fixed) lower bound. In the limit where $N \to \min\{|\mathscr{C}_1|, |\mathscr{C}_2|\}$, where $|\mathscr{C}_i|$ denotes the number of points in cluster $\mathscr{C}_i$, the domain of ξ approaches the point $i - c$ (see the right of Fig. 5.9). In such cases, choosing values of d such that

$$d \geq 0.8\,(i - c) \tag{5.4.7}$$

ensures that the two clusters will be merged with a relatively high probability (but not complete certainty). Of course, if one does not care about overestimations, (5.4.5) suffices in ensuring the merging of the two clusters irregardless of N, since the maximum upper bound of ξ can be obtained only in the case $N = 1$ and is equal to $c + o$.

Closing this exercise (and therefore the whole set of exercises), it is noted that if the requirement was to find with certainty *under what circumstances the ring and the circle will **not** be merged into a single cluster*, then the answer would be "as long as $d < 0.8\,(i - c)$ holds", for any given value of N, since the lower bound of ξ is constant in all cases.

References

[1] F Bodon, "A fast apriori implementation," *ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.

[2] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," *ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.

[3] C. Borgelt, "An implementation of the fp-growth algorithm," *ACM Press*, 2005.

[4] J. Leskovec, A. Rajaraman, and J. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2019.