Pantools: The effects of the *k*-value on the construction, storage and functionality of yeast pan-genomes

Raoul Timmermans

Bachelor thesis Bioinformatics
Raoul Timmermans
930703835100
Juni 2017

Begeleiders
S Sheikhizadeh Anari
prof.dr.ir. D (Dick) de Ridder

Coordinator
prof.dr.ir. D (Dick) de Ridder

Abstract

Motivation: This research focusses itself on Pantools, a tool that receives multiple genomes as input and constructs a generalized De Bruijn graph for the representation of pan-genomic data from these genomes. In these graphs, nodes represent a small segment of nucleotides of length *k*, called *k*-mers. We wanted to investigate the effect of k on the construction, storage and functionality of the pan-genome and if it is possible to get an optimal *k*-value. We also conducted a second experiment, looking into the differences between a newer version of PanTools and the older released version.

Results: We used two yeast genomes and were able to analyse the effects of the k-value on the pan-genomes as well as give an optimal range of k for the yeast genomes. We also found out that the newer version of PanTools is much more efficient than the old one, based on running times and database size.

Contents

1 Introduction

The number of genomes that represent many closely related phylogenetic groups and species can be quite numerous nowadays. The traditional way of comparing these genomes was to compare every single genome to a reference genome. That number of genomes can be quite large, this method is becoming inefficient. To cope with this increase in genomic information, a new approach was made, the pan-genome approach, originally intended to compare all of the genomes of a certain species, to find what the core genes and the dispensable genes are (Tettelin *et al.* , 2005). The term pan-genome has been expanded since then and now includes more ways of gathering information on different genomes. One of these methods is PanTools (Sheikhizadeh *et al.* , 2016), this tool makes use of a De Bruijn graph, which allows for the compression of multiple genomes and is the tool of interest for this research.

PanTools differentiates itself from other programs by making the compressed De Bruijn graph in a Neo4j (van Bruggen, 2014) graph database. The tool allows the input of multiple genomes and constructs a pan-genome out of these genomes. Neo4j makes a visual representation of the pan-genome, allowing a quick view of the genomes and their sequences. Neo4j also allows the use of Cypher (van Bruggen, 2014), which makes it easy to search the pan-genomes for properties and structures of interest.

Pantools starts by building a *k*-mer index, these *k*-mers are unique smaller parts of the sequences of the genomes. The length, i.e. the number of nucleotides, of these parts can be chosen and is called the *k*-value. The *k*-value not only influences the representation of the graph, but also the size of the database and the running time of the program. The problem however, is that it is unclear how exactly *k* influences these factors. Research is needed to see how *k* affects various factors for several datasets.

In this report, we study these effects of *k* on the performance of PanTools and the characteristics of the resulting pan-genome. By running PanTools for different *k*-values and for different yeast datasets, consisting of variable numbers of genomes, we want to see how the *k*-value influences the aforementioned factors. Two datasets will be used, one smaller dataset and one larger dataset. First the results of the smaller dataset will be analysed, which should give some valuable insights into the effects of *k*. These insights will make it possible to try and give an optimal *k* for the used dataset. The results of the larger dataset are compared to the first and if they show similar trends, a general solution for the *k*-value can be given. Finding this solution will help the program, optimizing the runtimes and the information that the graph gives.

A second research question concerns the difference between two version of Pantools. Since the beginning of this research, PanTools has evolved. The newest version of Pantools uses different techniques, which should make the program much faster than earlier versions. To see if this is the case, a comparison is made between the two different versions.

2 Methods

In this research, we focus on PanTools, a tool for representation of pan-genomic data. This program defines the pan-genome as a hierarchy of nucleotide, annotation and proteome layers stored in a Neo4j graph database. We will investigate the effect of *k* on the construction, storage and functionality of the pan-genome at the nucleotide layer.

2.1 Graph construction

PanTools receives multiple genomes as input and constructs a generalized de Bruijn graph (DBG) from these genomes, directly in a Neo4j database. In a DBG, nodes represent a unique couple of nucleotides of length *k*, called *k*-mers and edges indicate an overlap of *k-1* nucleotides between adjacent *k*-mers. In the generalized DBG used in Pantools, non-branching paths are compressed in one node. All nodes represent both forward (F) and reverse (R) sequences leading to four types of edges: FF, FR, RF, RR.

To construct the generalized DBG *k* needs to be specified. Then the construction starts with indexing all the *k*-mers by using KMC2 (Deorowicz *et al.* , 2014). After all the *k*-mers have been indexed and sorted the program starts to process all genomes subsequently. This processing is actually the construction of the form of the graph by four different operations: create, extend, follow and split. The sequence of each genome is scanned, one after another and when it encounters a *k*-mer that hasn't been visited before, a node of length *k* is created. This node is extended until reaching to a previously encountered *k*-mer . When this occurs, the *k*-mer in question is located and depending on its direction in the node, the node gets followed forward or reverse. After the follow operation, the split operation takes place, splitting a node into two nodes.

Figure 1 represents a very simple pan genome. It is clear that the SNP found in the middle of these two genomes is represented by bubbles in the graph. The branch length in the bubble is *2k-1*, since *k-1* *k*-mers span the SNP.

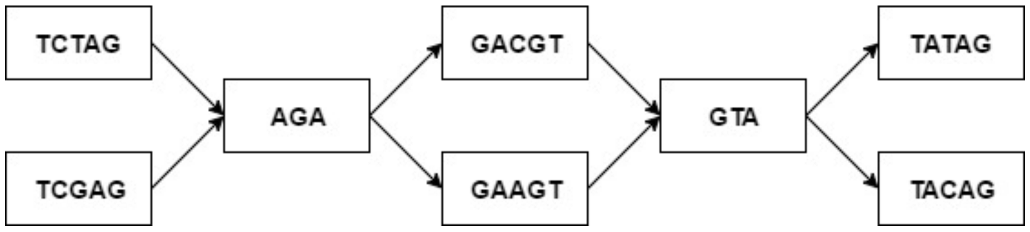

Figure 1. Simple pan-genome for two sequences, TCTAGACGTATAG(top) and TCGAGAAGTACAG(bottom). The first sequence has already been processed, *k*=3

2.2 The optimal *k*-value

The length of *k*-mers and is a very important factor for the creation of every DBG. Short *k*-mers are not unique and are repeated frequently in the input sequences. The number of *k*-mers increases until it reaches close the size of the data. Choosing a very small *k* will therefore result in an extremely tangled graph which is computationally hard to traverse. On the other hand, choosing *k* too large will result in a highly disconnected graph which is useless as a pan-genome, because it would not be able to capture the similarities between the genomes. This all shows that the *k* value should be chosen somewhere in between, where it avoids tangling the graph while keeping the graph as connected as possible. In the next sections we will show that the length of *k*-mers also influences other important factors, like the running time of the program, retrieval time and the database size.

2.3 Experimental setups

To see the effects of *k* on different factors we built pan-genomes of three and thirteen *Sacchromyces cerevisiae* genomes with a range of different *k* values, these pan-genomes will hence be referred to as pan-genome A and pan-genome B respectively. Table 1 details the factors of interest with a short description.

The idea is to get a good grasp of how different *k*-values affect the results. We chose odd numbers for *k*, because the odd numbers eliminate palindromes, which might affect the results. If the number is odd the centre nucleotide prevents the *k*-mer from having a palindromic reverse complement (Zerbino and Birney, 2008). The lower boundary of *k* in PanTools is 6, so the lowest possible odd number to start with, is *k*=7. The range should be big enough to get as much information as possible, so we chose *k*=35 as the higher boundary of *k* and took steps of 2 in the range of 7-35.

For these two datasets PanTools was run to construct databases of both sets for every *k*-value. The program presented results every time it completed its run and these were used to make graphs and draw conclusions. The runs were done in triplo for the running times, to get good average results. After these databases were made, another command of the program was used: genome retrieval, this retrieves the full genomes from the databases that were created. This command gives runtime as a result which also provides some insight into the effects of the *k*-value.

PanTools reports some information after every run, which was collected to make the graphs of this study and draw conclusions. The runs were done in triplo to randomize the experiments and get more reliable average results and were done on the dev1.ab.wurnet.nl server.

Table 1. The factors of interest

Property	**Description**
Number of *k*-mers	Number of *k*-mers
Number of nodes	Number of nodes
Number of edges	Number of connections between the nodes
Number of bases	Number of nucleotides in total
Graph size	Size of the total graph database in MB
Index size	Size of the index for the k-mers in MB
Processing time per genome	Time it takes to process per separate genome in seconds
Total processing time	Time it takes to process all genomes accumulative
Localizing time	Time it takes to add sequences to nodes
Total time	Processing time + localizing time
Genome retrieval time	Time it takes to retrieve the full genomes from the created database

The results can be grouped into three groups: 1) the graph characteristics, such as the number of *k*-mers, nodes edges and bases. 2) Size of the graph and index databases. 3) Construction and retrieval times.

2.4 Comparing two versions of Pantools

Besides the released version of Pantools, there is also a new version which uses just four types of edges (FF,FR,RF,RR) and stores the positional information on the edges instead of storing them in the nodes. To test the effect of this different design on some factors of interest (Table 1), the first version of Pantools was compared to the second version. In this experiment we constructed 9 pan-genomes for a range of 3-83 yeast genomes, with a stepwise inclusion of 10 genomes and the same *k*-values of 17. The runs were done on the altschul.bioinformatics.nl server.

3 Results

The goal of this study is to find a suggest of obtaining an optimal *K*-value that can be calculated for every dataset before starting the construction of the pan-genome. In this section we present the results on pan-genome A and then compare them to the results of pan-genome B.

3.1 Effects of *k*

First, we present the effect of *k* on the number of *k*-mers.

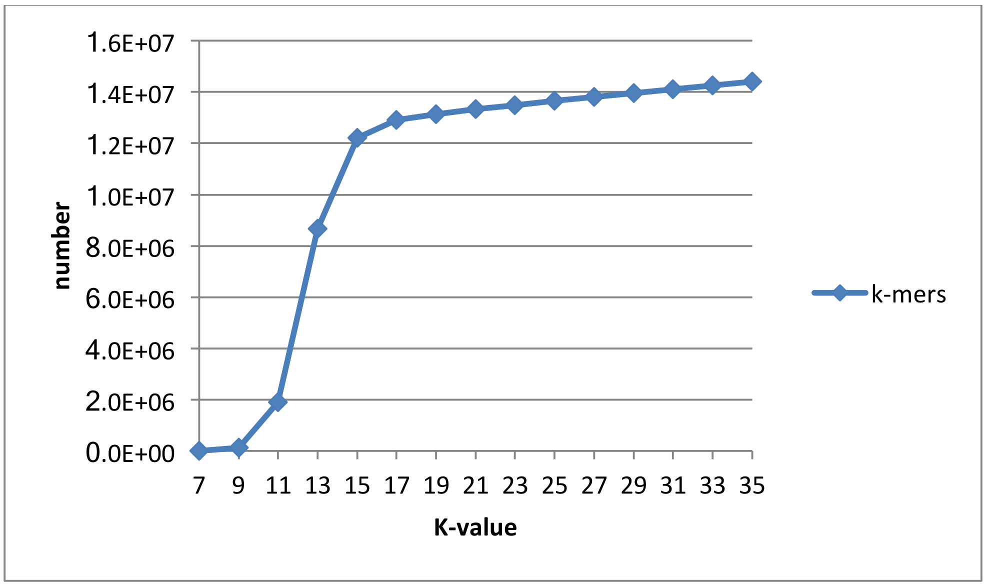

Figure 2. The number of *k*-mers, per *k*-value, for pan-genome A

Figure 2 shows that starting from two low values, the number *k*-mers suddenly goes up after *k*=9, before reaching to a smooth linear growth at *k*=17. The two initial values are exactly equal to the maximum number of canonical k-mers ($4^k/2$), which means that all the possible 7-mers and 9-mers are available in the data set. As the *k* increases, the data naturally has much more *k*-mers to offer, however at some point (here *k*=17) the growth trend switches from a steep linear growth to a smooth linear pattern. At this point the number of *k*-mers reaches to the maximum potential of the dataset, however there is still a smooth increase in the number of *k*-mers. The reason is that the *k*-mers coming from similar parts of the genomes, which are shared between sequences, start to become unique due to existing SNPs. To explain this effect consider the following example:

...TCGAGACGTAGAG...
...TCGAGAAGTAGAG...

These genomic regions are highly similar and there is one SNP in the middle. For *k*=3, there are 3 different k-mers spanning the SNP in each of the sequences, summing up to 6 different *k*-mers. By increasing *k* to 4 the total number of different *k*-mers which span the SNP position increase to 8. For large *k* values the total number of k-mers is expected to stand around the genome size. Figure 2 shows that at *k*=15 we reach to this number (12 million base pairs), as a result, from this point on the increase in the number of *k*-mers should be mainly related to short variations which exist between the genomes.

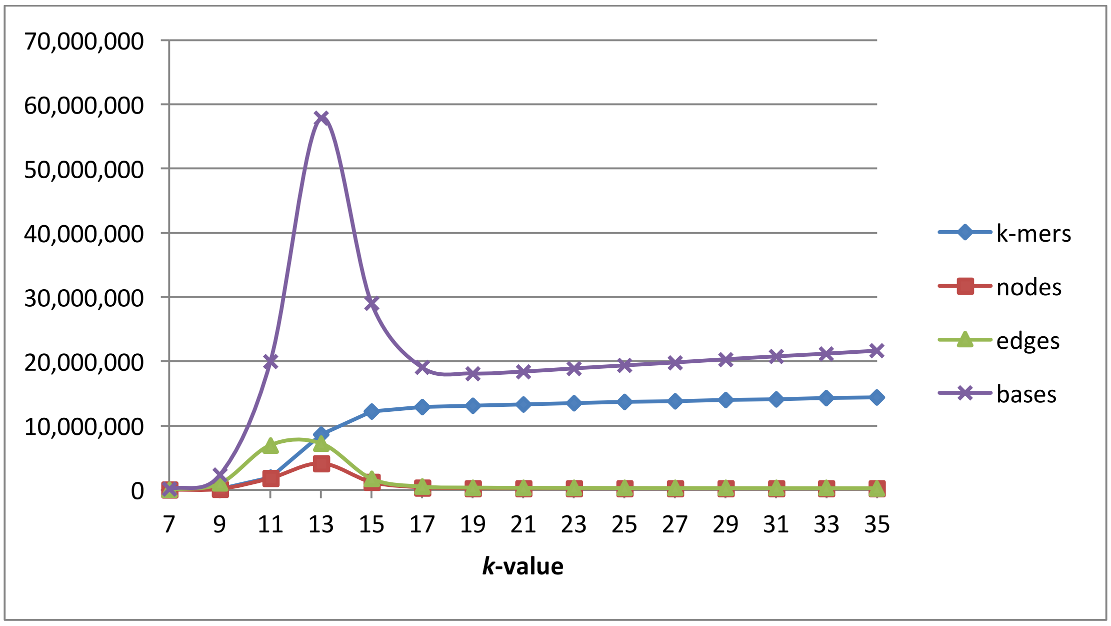

Figure 3. The number of nodes, edges and bases with respect to the *k*-value, in pan-genome A

Figure 3 shows that the trend of the changes in the number of nodes, edges and base pairs is highly similar. They all go up until they peak at *k*=13, after which they go down and take a linear growth format at *k*=17 onwards. The number of nodes is exactly equal to the number of k-mers for the first two points, but then it becomes lower than the number of *k*-mers, because non branching *k*-mers are compressed in a single node in the DBG. At the first two points all the *k*-mers are non-unique occurring in all the three genomes, as a result they are all branching in the graph and could not be merged. The number of nodes reasonably goes up as the number of *k*-mers increase, however it starts to fall after *k*=13, where it peaks. This is again related to the variations like SNPs between the genomes because SNPs can only be detected when they are less than *k* bases apart. Figure 4 shows that the SNPs between the two given sequences are captured in the form of bubbles. When *k* is set to 4 the sequences do not share *k*-mers anymore, resulting in two separate nodes for each genome.

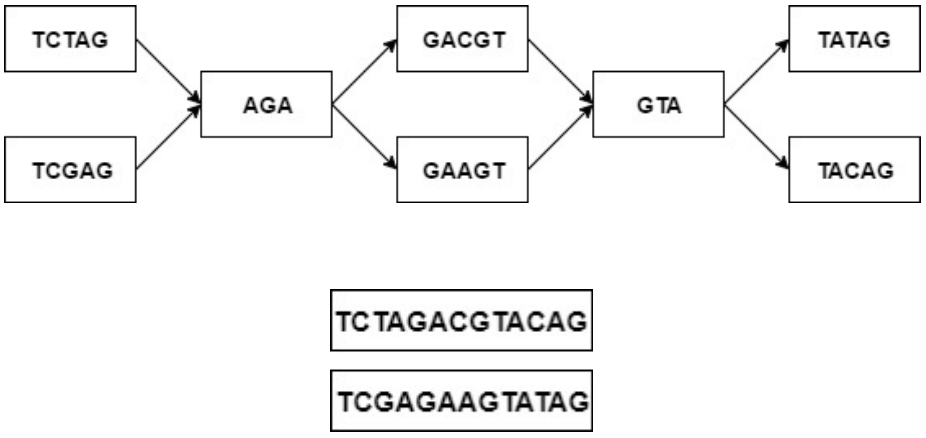

Figure 4. An example of how the *k*-value affects SNP detection, when constructing a database of sequences TCTAGACGTACAG and TCGAGAAGTATAG. For the top figure the *k*-value is set to 3, for the bottom it is set to 4

In the graph the number of nodes keeps decreasing as we increase *k* and the rate of this decrease is directly proportional to the percentage of the SNPs, which used to be less than *k* base pairs apart. So, from the graph we conclude that the number of SNPs which are 13bp apart is three times as large as those being 15bp apart. We also conclude that at *k*=13 the pan-genome is in its most informative state and captures the highest number of variations between the three genomes.

The variation in number of edges is expectedly in line with the variation in the number of nodes, because more edges are needed to connect the nodes. The main factor contributing to the number of base pairs stored in the pan-genome is the number of nodes; that is these two diagrams peak at the same point. However, from *k*=17 on, the number of base pairs increase steadily while we know that the number of nodes decreases very smoothly. The reason is that from this point on the majority of the captured SNPs are at a large distance (probably much larger than *k*) from one another and increasing *k* by 1, increases the length of these bubbles branches by 2.

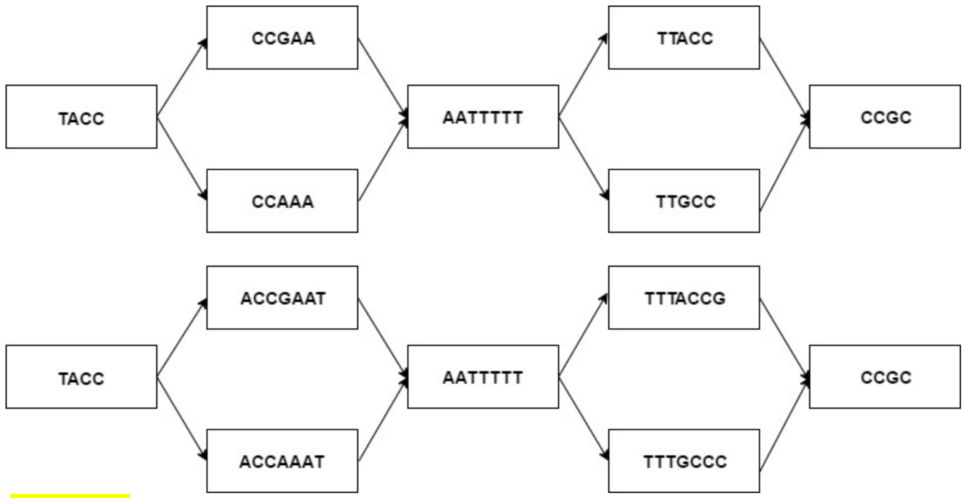

Figure 5. Two simple parts of a pan-genome graph for two sequences, TACCGAATTTTTACCGG and TACCAAATTTTTGCCGG. The *k*-value for the top figure is 3 and for the bottom it is 4

Figure 5 shows that for *k*=3 the number of base pairs is 35, but for *k*=4 there are 43 base pairs. Notice that the form remains the same and that there are four bubble branches, so the number of base pairs increases by 8 for *k*=4.

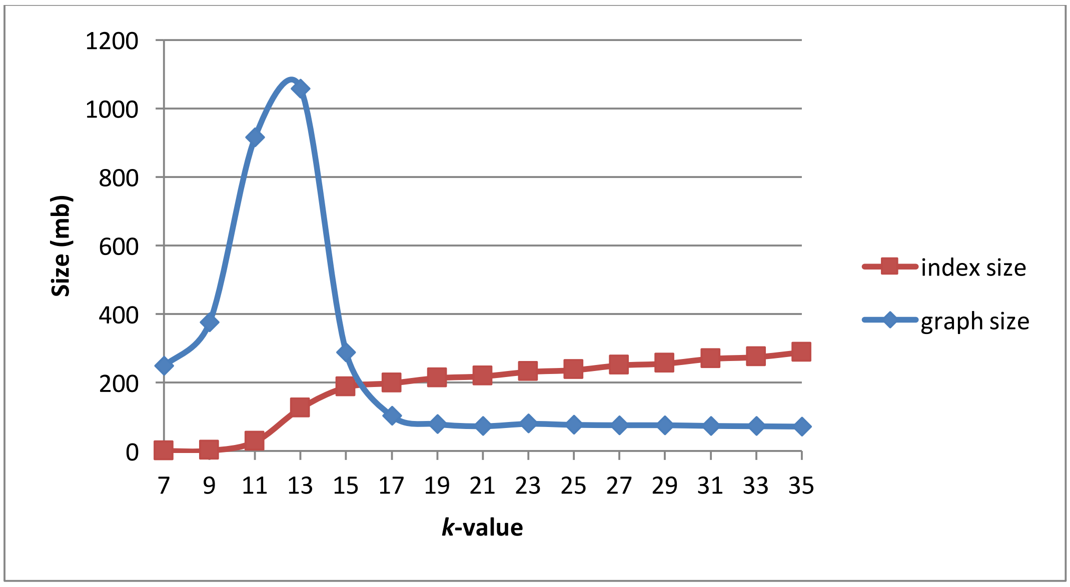

Figure 6. Database size and index size with respect to the k-value, in pan-genome A

Figure 6 shows that the database size follows a similar trend as the number of bases, edges and nodes. There are more of these characteristics, so the size of the database also increases with it. One noticeable thing here is that the size of the databases starts off high, indicating that there has to be another factor that influences the database size. These are the arrays in which information about the location of *k*-mers is stored, called the positional arrays. If *k* is small, the *k*-mers will be repeated at many positions, and every position is added to the positional array, resulting in a large starting size. The trend then follows the graph of the characteristics and at *k*=17 the database size starts to decrease steadily. This is the point at which the positional arrays are influencing the results again. Since the length and the uniqueness of the *k*-mers keeps increasing for each sub sequential *k*, less positions have to be added to the positional arrays and the database size will decrease accordingly. The figure also shows the index size, which follows the trend of the *k*-mers, since it indexes them.

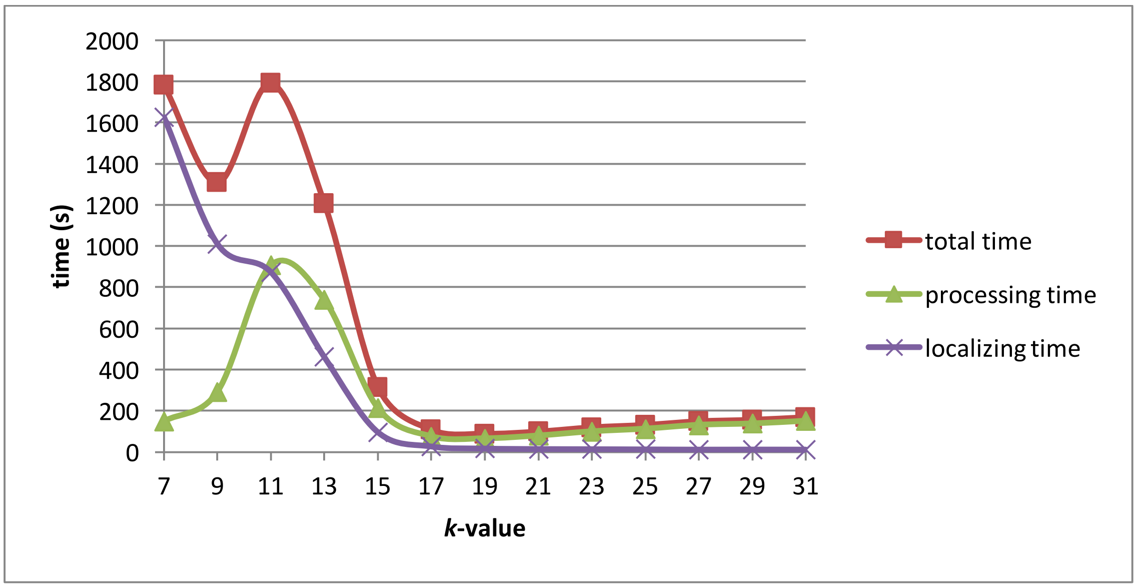

Figure 7. Average total, processing and localizing time with respect to the *k*-value, in pan-genome A

The total time consists of the processing time and the localizing time. The average times from three runs are shown here and since the standard deviations were so small, there are no error bars included in the graph. The localizing time is the time it takes to localize all the positions of the indexed *k*-mers and to add their sequence to the nodes. What's noticeable here is that the sequencing time starts off very high and goes down to almost zero and that the processing time peaks at *k*=11, which is different from the rest of the graphs. After *k*=*17,* the localizing time almost goes to zero and the total time then only depends on the processing time. The high localizing time at the start can be explained by the same reason of the high start for the database size. At small k-values are repeated many times and all these positions have to be localized, resulting in a long localizing time. As the k-mers become bigger and more unique, the localizing time goes down, continuing up until *k=17*, where the k-mers have all become unique so the localizing time becomes steady. To explain the processing time we have to take a closer look at the processing time per genome.

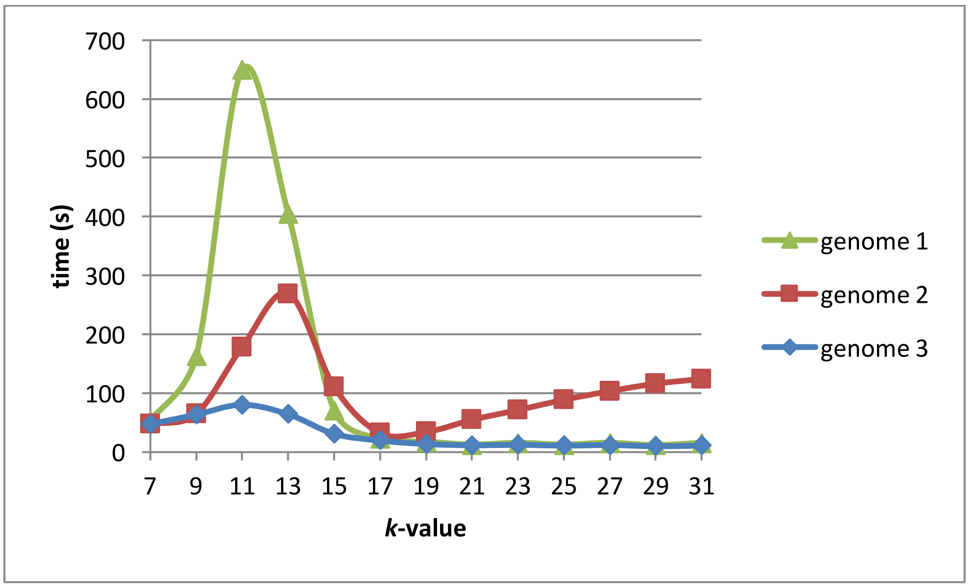

Figure 8. Processing time per genome with respect to the k-value, in pan-genome A

Figure 8 shows very different trends for every genome. The first genome has a large peak at *k*=11, the second has a large peak at *k*=13, and the third remains relatively steady. All the genomes become steady after *k=17*, genome 1 and 3 remain stable, but genome 2 grows in a linear trend. For the first genome the *k*-mers are still repeated at *k*=11 and therefore many splits have to made, which is the most time consuming operation. At *k*=13 the k-mers are more unique and will not be repeated as much, so the time goes down. This continues up until *k*=17, where the number of splits reaches its minimum, because all the *k*-mers have become unique. The second genome has a peak at *k*=13, because many splits have already been made at *k*=11. After *k*=13 the time decreases, corresponding to the number of nodes, bases and edges, but at *k*=17 it starts to rise linearly. Since the two genomes are now compared, the SNPs between the two genomes are detected. We saw earlier that the number of bases and *k*-mers keep increasing after *k*=17, due to the SNP detection, so there's increasingly more information to process with each increasing k, resulting in high processing times. The third genome shows that it is relatively, because the yeast genomes similar and most of the SNPs have already been detected in the comparison between the first and the second genome. For the third genome most of the splits has already been made, so not much time has to be spent on the third genome.

3.2 Comparison pan-genomes

In this segment we take a look at the compare the results of pan-genome A and pan-genome B.

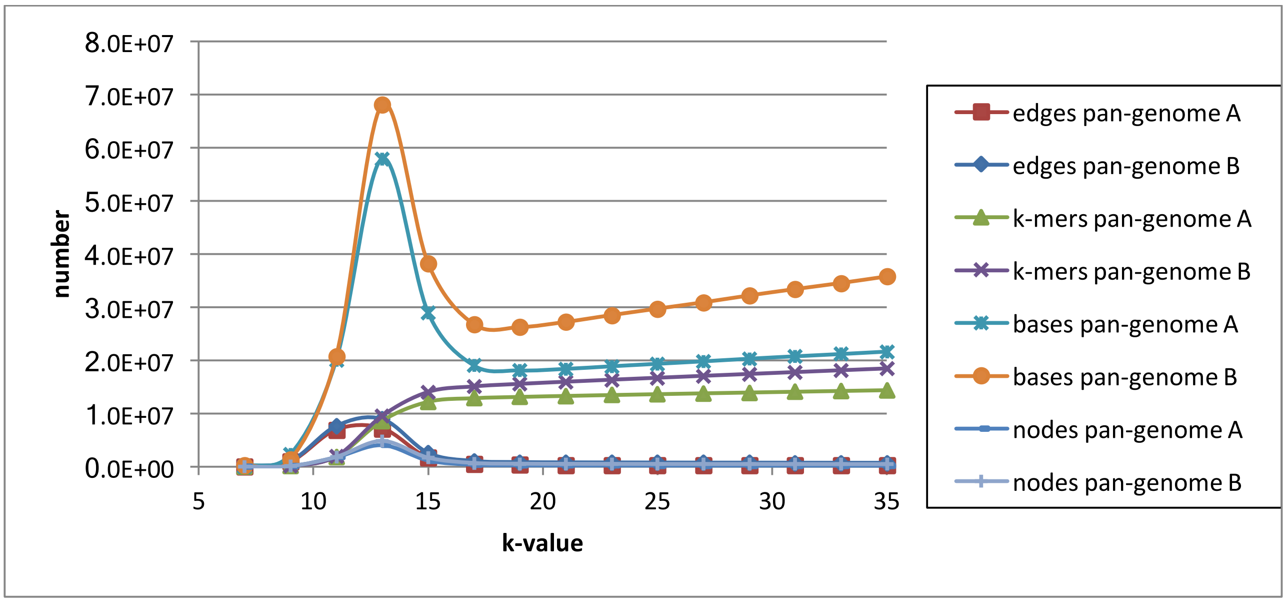

Figure 9. Number of nodes, edges bases with respect to the k-value, in pan-genome A and B

Figure 9 shows that all the characteristics of the graph have generally similar trends, the difference is that the values for pan-genome B are generally higher. The first two points of the *k*-mers and nodes are exactly the same, since the *k*-mers are all non-unique at these points. The third point also shows large similarities, but from *k*=13 onwards there is a difference, pan-genome B’s results are much higher, because more SNPs are detected between the thirteen genomes. This also causes the steep increase for the *k*-mers and bases in pan-genome B, more SNPs indicate that there are more branches that are lengthened with each increasing *k* and it also means that there are more *k*-mers that span these SNPs. The graphs for the running times also showed highly similar trends, the running times were higher for pan-genome B, but the form of the graphs were the same. The database does show some variations though.

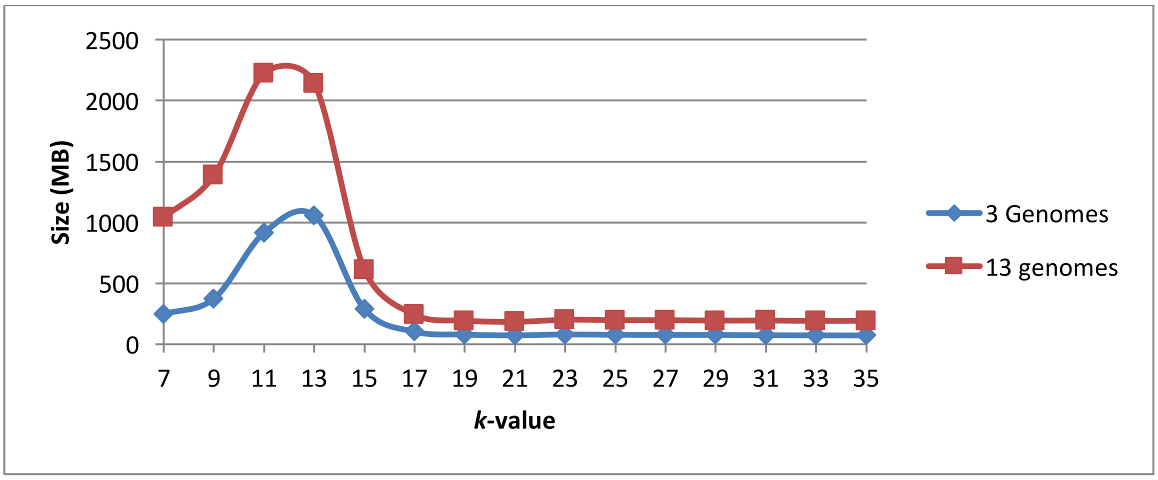

Figure 10. Graph database size with respect to the k-value, in pan-genome A and B

Figure 10 shows that the trends stay similar with one exception being that the peak of pan-genome B shifts to the left for the 13 yeast genome dataset. This can be explained by looking back at the processing time per genome and the positional arrays. The processing time showed that for the first genome there was a very large peak at *k*=11, which also holds true for pan-genome B. While the *k*-mers were already being repeated in the smaller dataset, the number of repetitions will be much larger for a larger dataset, resulting in larger positional arrays. Since this peak doesn't coincide with the number of modes, bases and edges, it shows that the positional arrays have a much higher impact on the larger dataset. One other thing that is different here is that the line after *k*=17 doesn't seem to decrease, while we just found that this should be the case. The line does decrease, but because the genome size is much larger this effect will be much slower than with a smaller dataset.

3.3 Comparison Pantools

The other part of this research was the comparison between the first and the second version of PanTools, which are discussed here

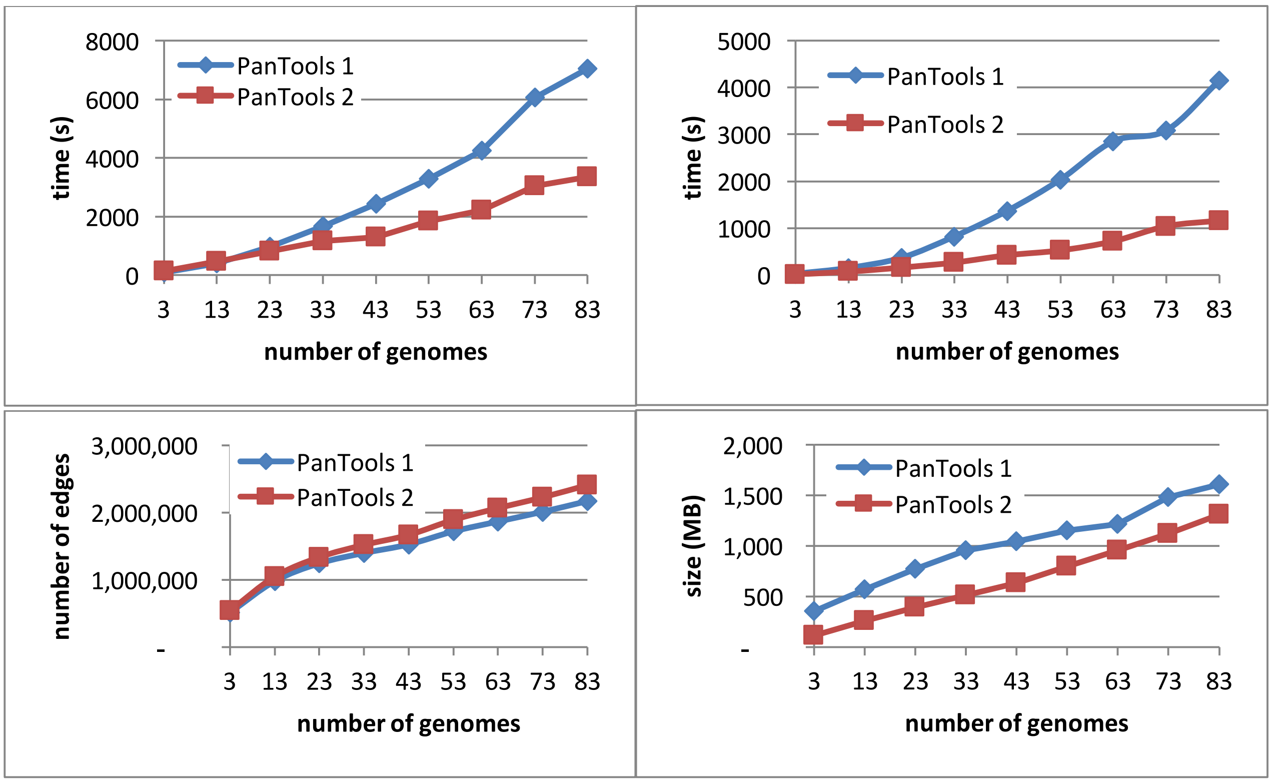

Figure 11. Comparison between the two version of Pantools for different number of genomes. The runtime for the construction of the database is compared(top left), the retrieval time (top right), the number of edges (bottom left) and the size of the database(bottom right)

Figure 11 shows that for the database size and the time, PanTools 2 is much more efficient. The construction time for the first three points is lower for PanTools 1, which is related to the use of an unstable server, but also to the number of edges being very similar for the first three points and after these points PanTools 2 is much faster. The database size is much larger for the first Pantools than for the second version of Pantools from the beginning. This has something to do with the positional arrays, but we are not exactly sure what. The only thing that Pantools has higher values in is in the number of edges. The information of the direction is stored on the edges instead of the nodes, resulting in more edges. The difference isn't that big as there are overall around 10% more edges for Pantools 2 than Pantools 1. The retrieval time shows a substantial difference between the two version as the time becomes four times as big for the first Pantools at 83 genomes. The second version allows the program to search for only the edges and not go through the properties of the node, it is much quicker for such an action.

4 Conclusion

Now that it's known how the k-value effects the various results and how different datasets compare to each other, we can take a look at the optimal *k*. The results from pan-genome A showed that most of the results share a peak at *k=13* and a plateau after *k=17*. The values before the peak will contain a lot of very small connected nodes explained earlier and after the plateau all of the k-mers will be unique resulting in the large disconnected nodes. This means that there is such a thing as the optimal *k* and it should be in the range of 13-17 for this dataset. At k=13 the pan-genome contains the most information and at *k*=17 all the k-mers have become unique, resulting in disconnected nodes. The choice of *k* depends on where the focus lies for the results. The focus here lies on the running time for the program, which should be fast, and the size of the database, which should be small. For this goal the best option of *k* here would be 17, since this is the point at which the running times are the fastest, the database size is the smallest, but not all the *k*-mers have yet become unique, ensuring that a lot of the variation between the genomes is still found.

Armed with the knowledge that there actually is such a thing as an optimal *k*-value, we now wanted to know if the solution for one dataset can be made into a general solution. The comparison between pan-genome A and B showed that the graphs followed the same trend, which means that the general solution can be made for these datasets. Now that this is known, the solution for pan-genome A has to be "rewritten" into a solution that can be obtained by analysing the trend of the graphs. If such a solution can be found, Pantools itself can look for a certain point where it can set the optimal k. The best result to look at for the optimal *k* is the number of k-mers. For pan-genome A, *k=17* is the point at which the k-mers start to flatten. So this means that as a general solution, the program can be set to look at the first point where it can see the plateau of the k-mers start, which is at *k*=17.

While we have found general solution for the optimal k-value for these datasets, we don't have enough evidence to say that this holds true for every dataset that can be used. Repeating the experiments for the 93 yeast genome set will verify all the assumptions that we have made here, but more research is needed than just the yeast genomes. By repeating this experiment with more different species, for example brassica plants, it can be verified if the assumptions made in this research are true for other datasets and if this really can be used as a general solution and not just as a solution for the yeast. Another thing that should be investigated more thoroughly is the SNP detection for every *k*-value. The number of SNPs can be found in Neo4j and doing this for every *k*, will give much more insight in the detection of the SNPs and since much of this research is based on this detection, it would be nice to have a better understanding at how the number of SNPs changes.

Our other research question was if PanTools 2 is better than PanTools 1. Since both the construction times and the genome retrieval times were much lower for the second version and the databases are much smaller, it's safe to say that it is.

5 References

Deorowicz, S., Kokot, M., Grabowski, S., & Debudaj-Grabysz, A. (2015). KMC 2: Fast and resource-frugal k-mer counting. *Bioinformatics*, *31*(10), 1569-1576.

Sheikhizadeh, S., Schranz, M. E., Akdel, M., de Ridder, D., & Smit, S. (2016). PanTools: representation, storage and exploration of pan-genomic data. *Bioinformatics*, *32*(17), i487-i493.

Tettelin, H., Masignani, V., Cieslewicz, M. J., Donati, C., Medini, D., Ward, N. L., ... & DeBoy, R. T. (2005). Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: implications for the microbial "pan-genome". *Proceedings of the National Academy of Sciences of the United States of America*, *102*(39), 13950-13955.

Van Bruggen R. (2014). Learning Neo4j. Packt Publishing Ltd , Birmingham, UK .

Zerbino, D. R., & Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, *18*(5), 821-829.