Notebook: nanogate

AngeloGaeta90

November 17, 2020

Contents

1 Introduction

1.1 Changelog

1.2 v0.0.0 - Initial release

cript **testbeds generator.py** to generate testbeds implemented script `/nanogate_project/data/utils/proto_nanogate.py` implemented applying naive algorithm

v0.0.1 - testbed generator output 08/11/2019 script `/nanogate_project/src/nanogate_project/pre_processing/testbeds_generator.py` now generates as output 1 fasta file for all constructs, 1 fasta file for all parts , 1 .csv table to map constructs and parts

script **nanogate.py** implemented

Snakefile snakemake pipeline implemented

1.3 v0.0.2 - jaccard threshold kmer length 15/11/2019

1. nanogate tool has been moved out from the repo
2. the script **process table.py** has been implemented to process nanogate results
3. the scripts **jaccard evaluation table.py**
   and `/nanogate_project/src/nanogate_project/post_processing/jaccard_evaluation_table.py`
   have been implemented to create table and plot related to the jaccard threshold value
4. the snakemake pipeline has been updated with new rules

1.4 0.0.3 - pipeline-fixes and poisson filter 29/11/2019

1. the folder structure odf the project has been reshaped, no command line scipt is required to perform a full analysis
2. A poisson filter has been applied to cut out reads too small
3. true and false positive and summary stats added

1.5 v0.0.4 - nanogate filtered and row, filter by sampling, truepositive fix 06/12/2019

1. Nanogate is now split into two different methods raw and filtered: raw evaluates the Jaccard for each part filtered filter out parts on the basis of constructs length and jaccard value
2. theoretical samples are now generated by sampling and assembling parts until their average is stable
3. True positive are now evaluated per simple construct not by part

1.6 v0.0.5 - find kmer size given error rate, testbed similarity

1. In **nanogate** the kmer length is no longer required, can be evaluated from the error rate

2. `testbed_generator.py` now allow to define constructs similarity, the script need reshaping though

3. Snakemake pipeline now include error rate parameter

1.7 v2.0.0 - runs,time plot, stats per run, cloned parts

1. the pipeline now allows the execution of multiple runs

2. Cloned parts are not considered during the evaluation of the error rate

3. `summary_stats.txt` is now a table, called `run_stats.csv`

4. summary stats evaluate on all runs are now available

1.8 v3.1.0 - new filter method, clone list .json , error within error, mistmatch per error, short reads per error

1. the distribtution of constructs did not match a poisson distribution as consenquence
   now all values ¡ 5 percentile of the distribution are not considered now

2. the list of clone parts included into a json file are now
   used as input in **nanogate**

3. new stats after each run are now considered
   $correctparts/allparts$,
   $wrongparts/badmatches$,
   $mismarch/badmatches$,
   $missingparts/badmatches$

4. in **nanogate** now distribution of discarded reads, accepted reads and theoreticalreads are now plotted in output

5. constrcuts with more than 10 reads and average number of reads per constructs are now considered

1.9 v3.1.3 - good reads analysis 31/01/2020

1. The reads id taken from badreads is now in filtered nanogate results

2. The analysis of good reads per nanogate results and **accuracy** = good reads matches : good reads

3. the pipeline is now too slow though, need optimisation

4. nanogate methods draft

1.10 v3.1.3 - 70% identity analysis 7/02/2020

1. analysis time reduced
2. plots updated to include depth paramter
3. good reads truepositive , good reads and good redas percentage plots added
4. nanogate methods draft

1.11 v3.2.1 - good read criteria 7/02/2020

1. considering good reads reads with length at least 99% of the original constrcut

1.12 v.3.4.0

1. minimap2 added to the pipeline
2. added script to turn alignment to csv
3. added script to process alignment results
4. added script to evalute minimap2 results

1.13 v.4.0.0

1. minimap2 Real time and CPU time
2. parts in testbeds are no longer assemble at random, each part is assigned to a position before the assembly
3. added script to perform parts combinatorial assembly
4. minimap2 mapping quality analysed
5. minimap2 alignment score analysed
6. pipeline adapted to work on combinatorial assembly
7. scripts to write comparison tables and plot implemented

1.14 v.4.5.0

1. pipeline adapted to work with bad quality reads
2. nanogate adapted to work with bad quality reads

1.15 v.4.6.0

1. nanogate aggregation raw and filter nanogate are no longer divided
2. nanogate parts order with minimap2 approach for sorting parts implemented
3. nanogate analysis per constructs implemented
4. minimap2 analysis per constructs implemented
5. consufions matrix analysis for both nanogate and minimap2

1.16 v.5.0.0

1. nanogate parts order applying mappy approach implemented
2. nanogate parts order ratio, precision and recall analysis implemented
3. analysis of constructs with more than 10 reads implemented

2 Methods

2.1 Nanogate Pseudocode

- R = reads
- P= parts
- e= error rate
- jt= jaccard threshold
- pc= parts per construct

2.2 Snakemake pipeline

3 Results

4 V2.0.0

This simulation can be executed using the following snakemake config file:

```
project: "test"
genbank: "yeast_chromosome_xiv.gb"
run: 10

generate_testbeds:
    cds_number : 50
    constructs_number: [10,50,100]
    parts : [5,10,20]
    clone_parts_amount : 0.4
    parts_similarity : 0.7

badreads:
    quantity : "50x"
    error_model: "nanopore"
    qscore_model: "nanopore"
    glitches: "0,0,0"
    junk_reads: 0
    random_reads: 0
    chimeras: 0
    identity: 95,100,4
    start_adapter_seq: '␣""␣'
    end_adapter_seq: '␣""␣'
```

Algorithm 1 Nanogate

```
procedure NANOGATE($R, P, E, jt, pc$)
    $O \leftarrow$ RAW NANOGATE($R, P, E, k, pc$)
    FILTER NANOGATE($O, jt$)
end procedure
procedure RAW NANOGATE($R, P, E, k, pc$)
    if E then
        $K \leftarrow$ getkmer( $E$)                                ▷ Find kmer length given error rate
    else
        $K \leftarrow k$
    end if
    $PH \leftarrow$ Initialise()                                          ▷ list of parts
    for Pi in P do
        $PHi \leftarrow$ gethashes(Pi,K)
        $PH \leftarrow$ add(PHi)
    end for
    $Rf \leftarrow$ poissontest(R,pc)              ▷ given parts generate constructs length
distribution, and discard all reads too short to be part of the distribution
    $O \leftarrow$ initialise()                                        ▷ Initialise output
    for Rfi in Rf d do      ▷ compare each read against each part and return the
jaccard containment coefficient
        $RHi \leftarrow$ gethashes($RKi, K$)
        for part in P  do
            $J \leftarrow$ jaccardcontainment(PH,RHi)
            if $J > jt$ then
                $J \leftarrow$ addtoresult(O)
            end if
        end for
    end for
end procedure¶
procedure FILTER NANOGATE(O,jt)  ▷ For each read consider only the part
with the highest jaccard, and which sum is $\geq$ than read length
    for Rf in O do
        $Rfi \leftarrow$ jaccardfilter(O)
    end for
end procedure
```

```
nanogate:
        error_rate  :  ["001","010","030"]
        jaccard_thresholds: ["00", "01", "02", "03", "04",
        "05", "06", "07", "08", "09", "10"]
```

4.1 True positive

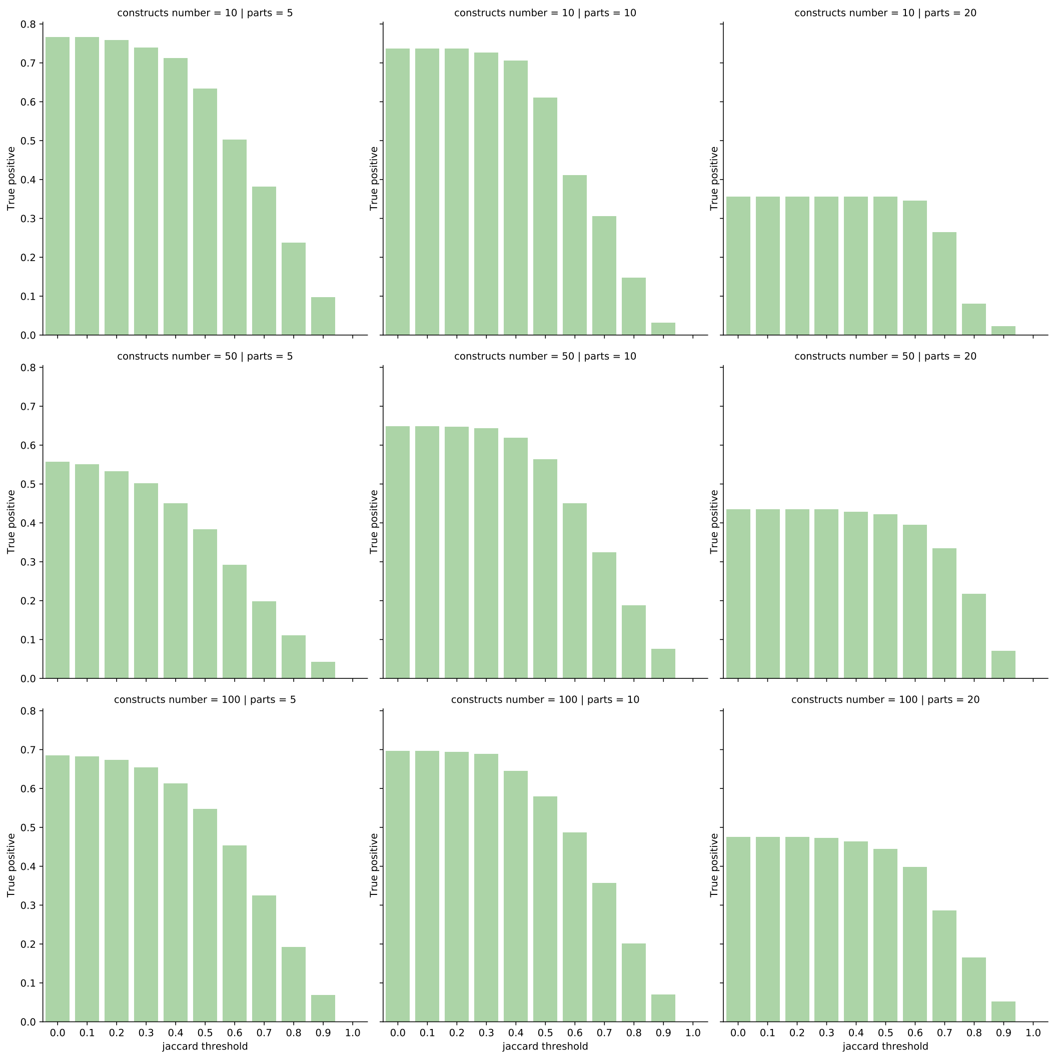

Figure 1: **001 Error rate run 0.**

4.2 Time plot

5 Results

5.1 V3.1.0

This simulation can be executed using the following snakemake config file:

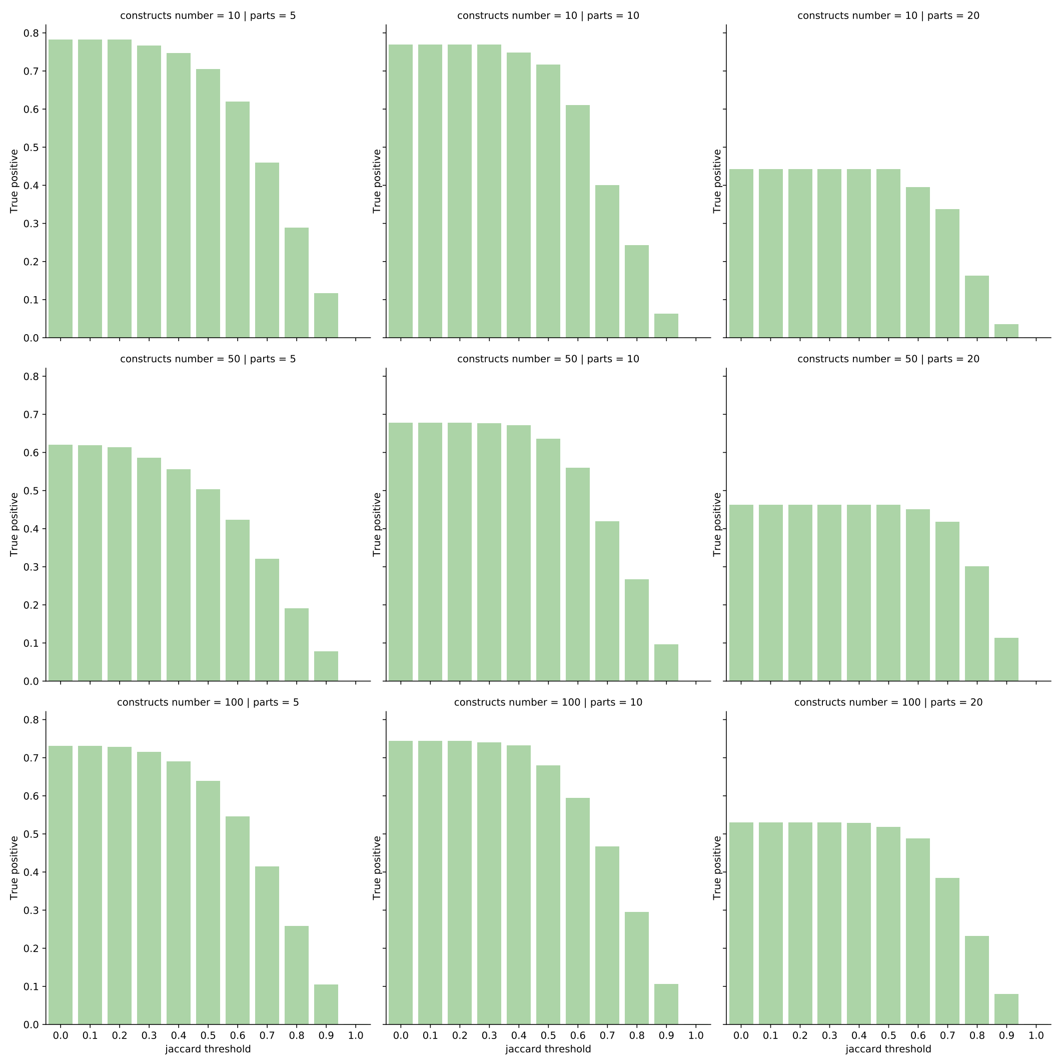

Figure 2: **010 Error rate run 0.**

```
project: "percentile_5"
genbank: "yeast_chromosome_xiv.gb"
run: 10

generate_testbeds:
    cds_number : 50
    constructs_number: [10,50,100]
    parts : [5,10,20]
    clone_parts_amount : 0.4
    parts_similarity : 0.7

badreads:
    quantity : "50x"
    error_model: "nanopore"
    qscore_model: "nanopore"
    glitches: "0,0,0"
    junk_reads: 0
```

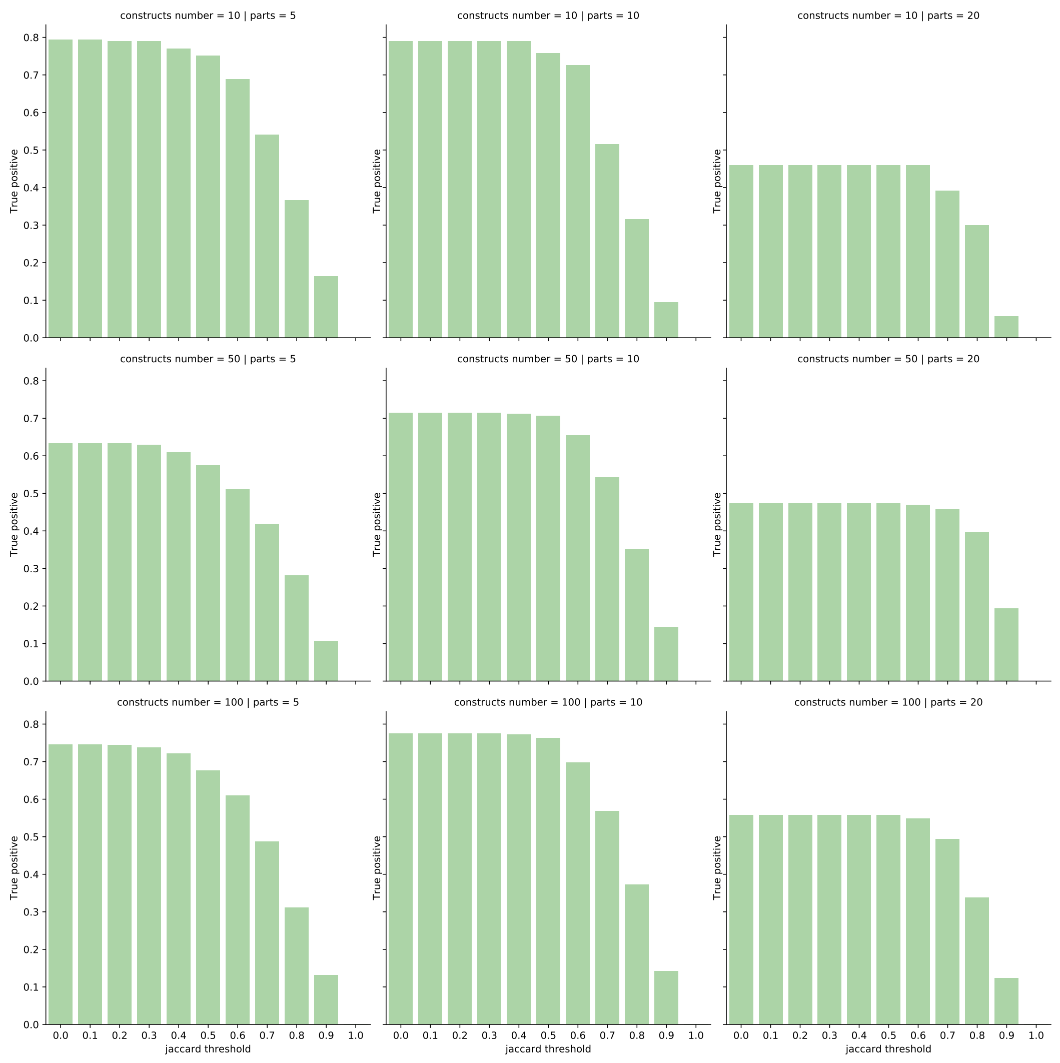

Figure 3: **030 Error rate run 0.**

```
    random_reads: 0
    chimeras: 0
    identity: 95,100,4
    start_adapter_seq: '␣""␣'
    end_adapter_seq: '␣""␣'

nanogate:
     error_rate  : ["001","010","030"]
     jaccard_thresholds: ["00", "01", "02", "03", "04",
      "05", "06", "07", "08", "09", "10"]
```

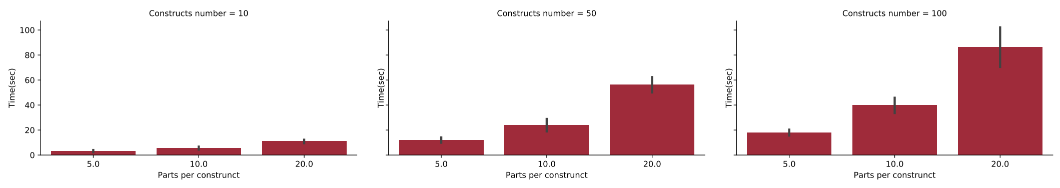

Figure 4: **Time plot**

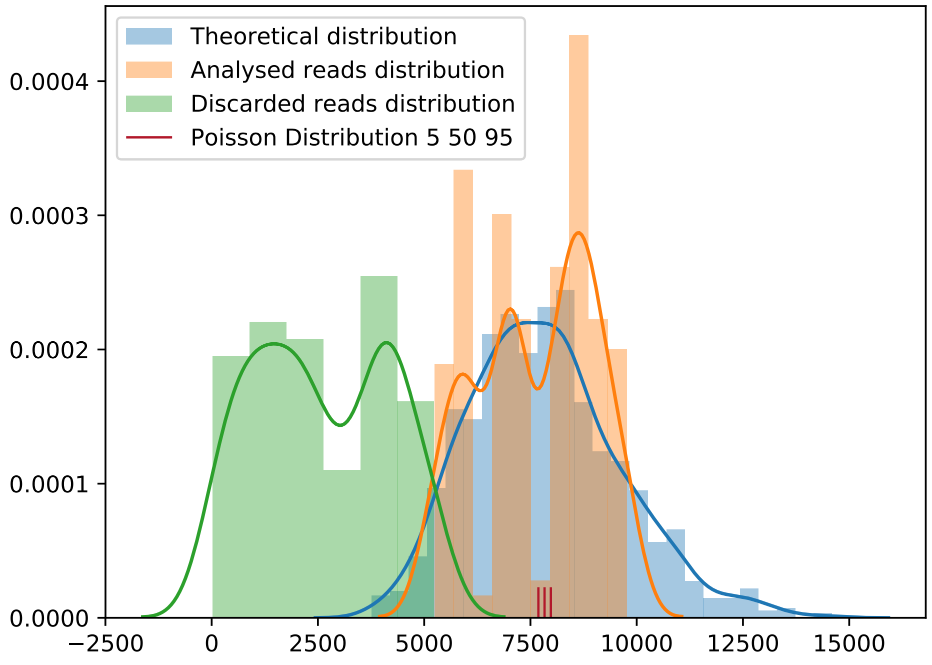

Figure 5: **5 parts reads distribution, the accepted reads now fits the theoretical distribution.**

5.1.1 Reads distribtution

5.1.2 Error within error

5.1.3 mismatch per error

5.1.4 short read per error

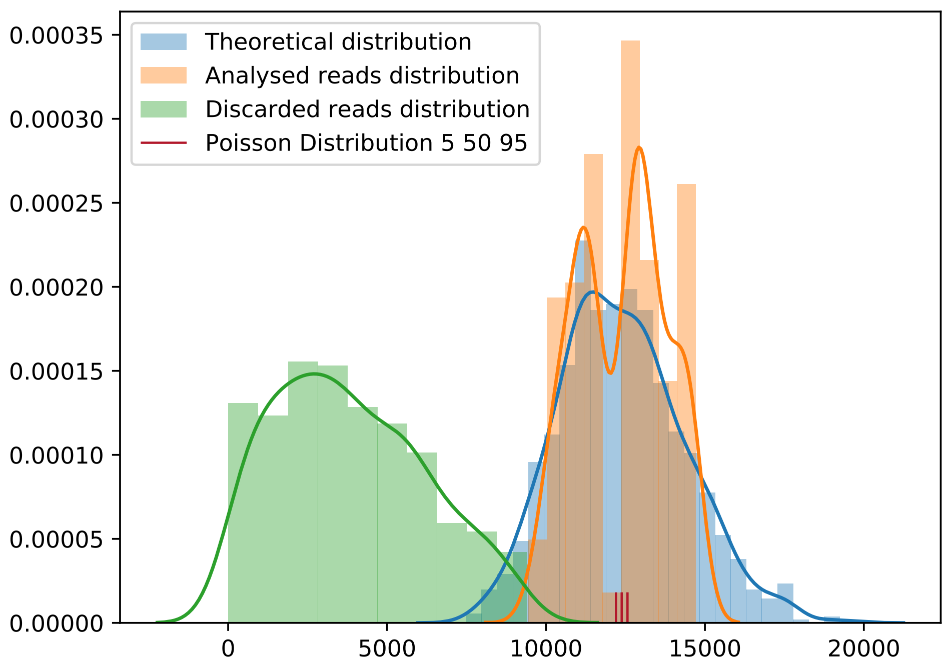

Figure 6: **10 parts reads distribution, the accepted reads now fits the theoretical distribution.**

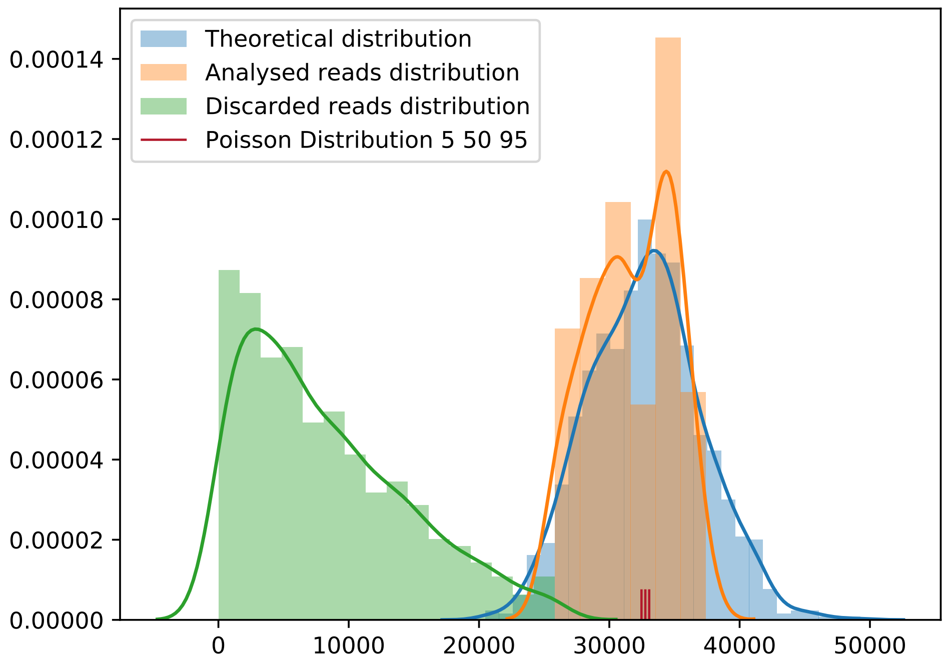

Figure 7: **20 parts reads distribution, the accepted reads now fits the theoretical distribution.**

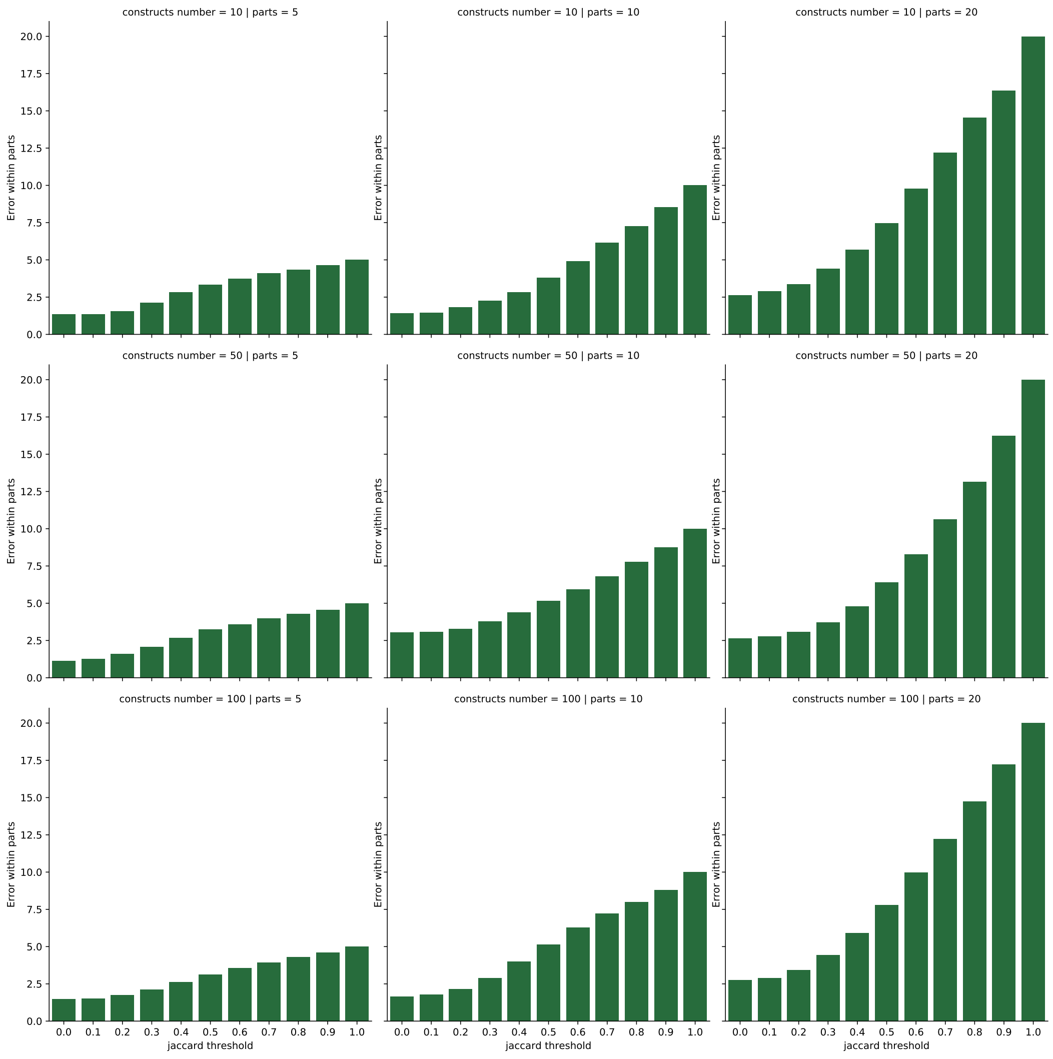

Figure 8: **Example of a Error within error plot taken from run 0.This metric is measured as** $missingorwrongparts/badmatches$

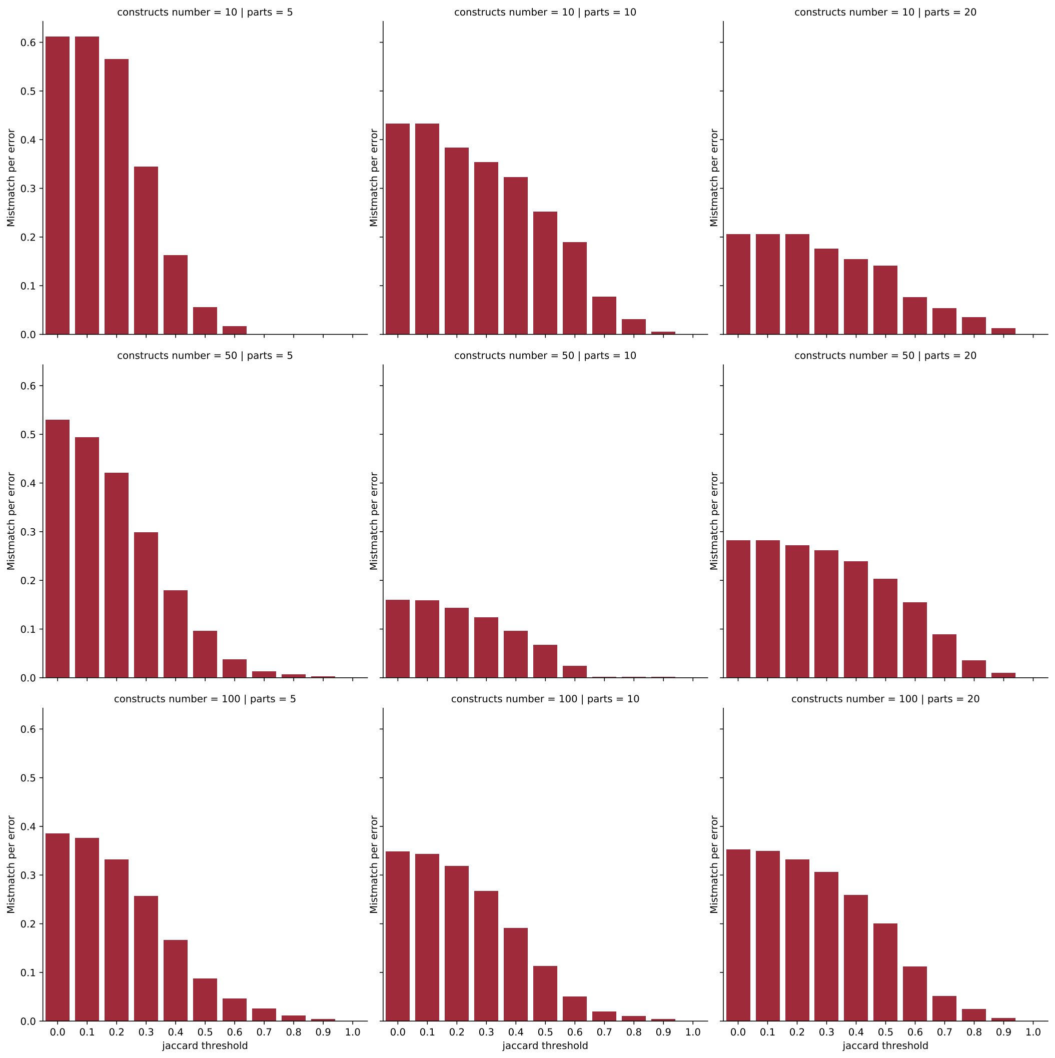

Figure 9: **Example of a mismatch per error plot taken from run 0. This metric is measured as** $readswithwrongparts/badmatches$

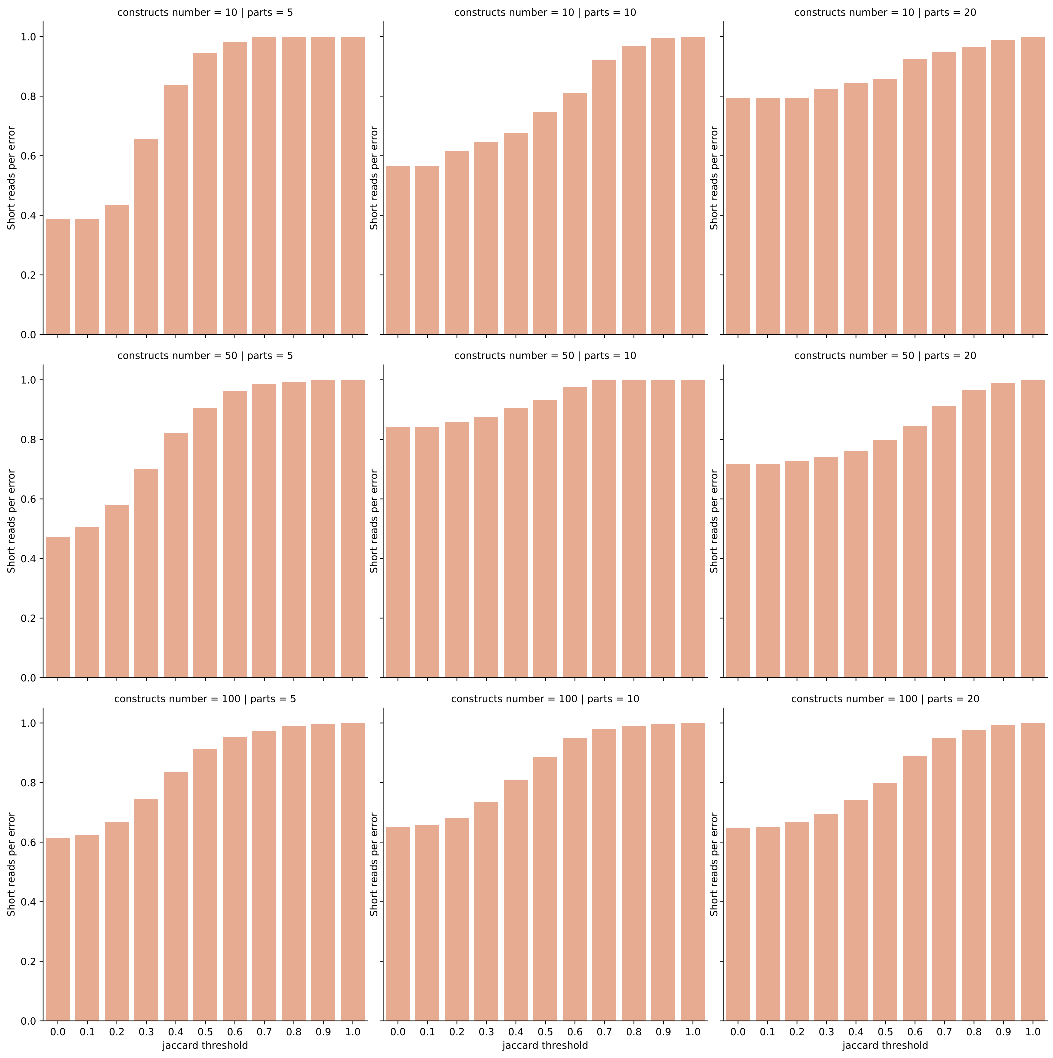

Figure 10: **Example of a mismatch per error plot taken from run 0. This metric is measured as** $readwithmissingparts/badmatches$

5.2 V3.1.3

This simulation can be executed using the following snakemake config file:

```
project: "percentile_5"
genbank: "yeast_chromosome_xiv.gb"
run: 1

generate_testbeds:
    cds_number : 50
    constructs_number: [10,50,100]
    parts : [5,10,20]
    clone_parts_amount : 0.4
    parts_similarity : 0.7

badreads:
    quantity : "50x"
    error_model: "nanopore"
    qscore_model: "nanopore"
    glitches: "0,0,0"
    junk_reads: 0
    random_reads: 0
    chimeras: 0
    identity: 95,100,4
    start_adapter_seq: ' "" '
    end_adapter_seq: ' "" '

nanogate:
    error_rate : ["001","010","030"]
    jaccard_thresholds: ["00", "01", "02", "03", "04",
    "05", "06", "07", "08", "09", "10"]
```

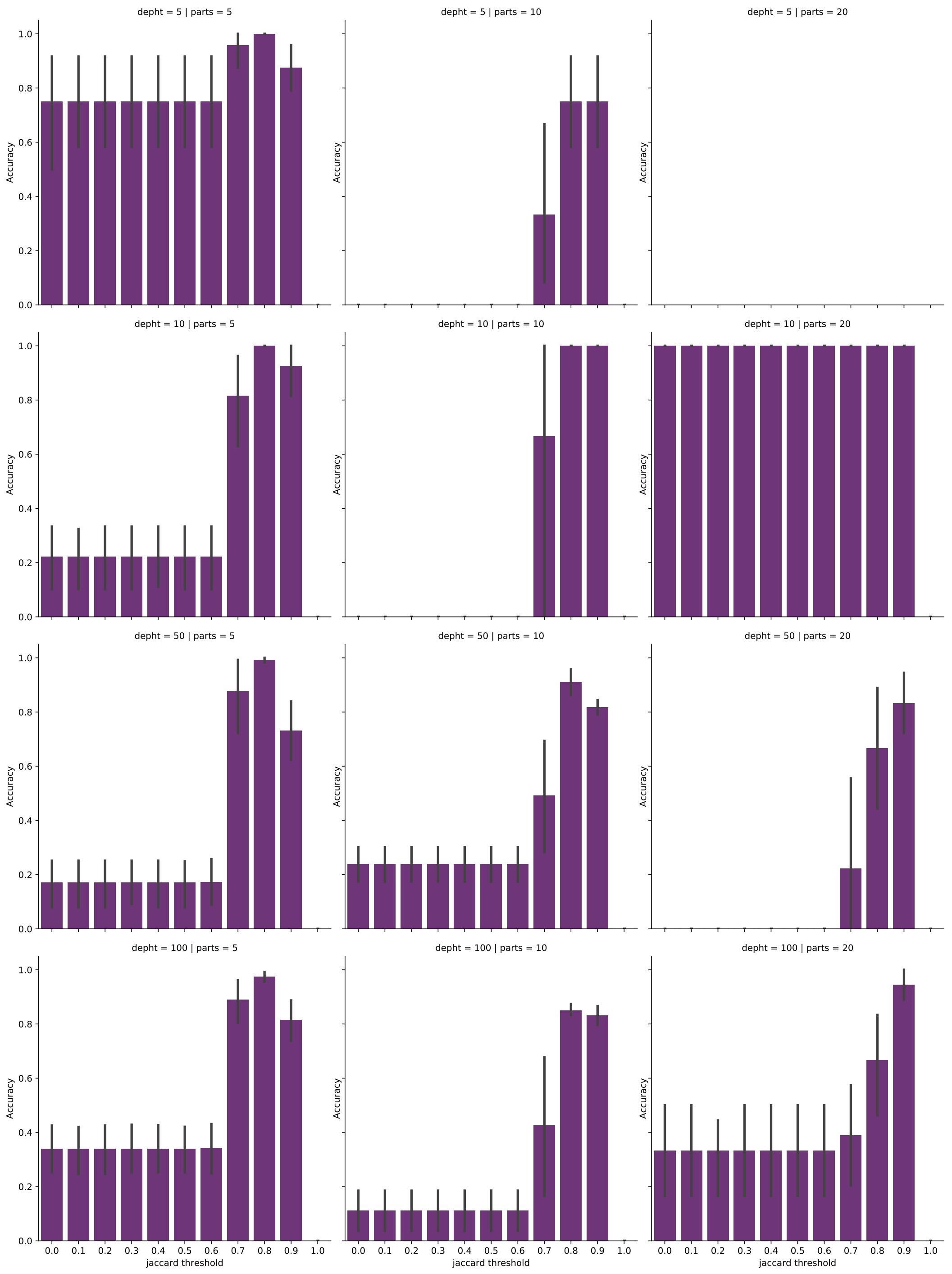

Figure 11: **True positives analysed only on good reads**

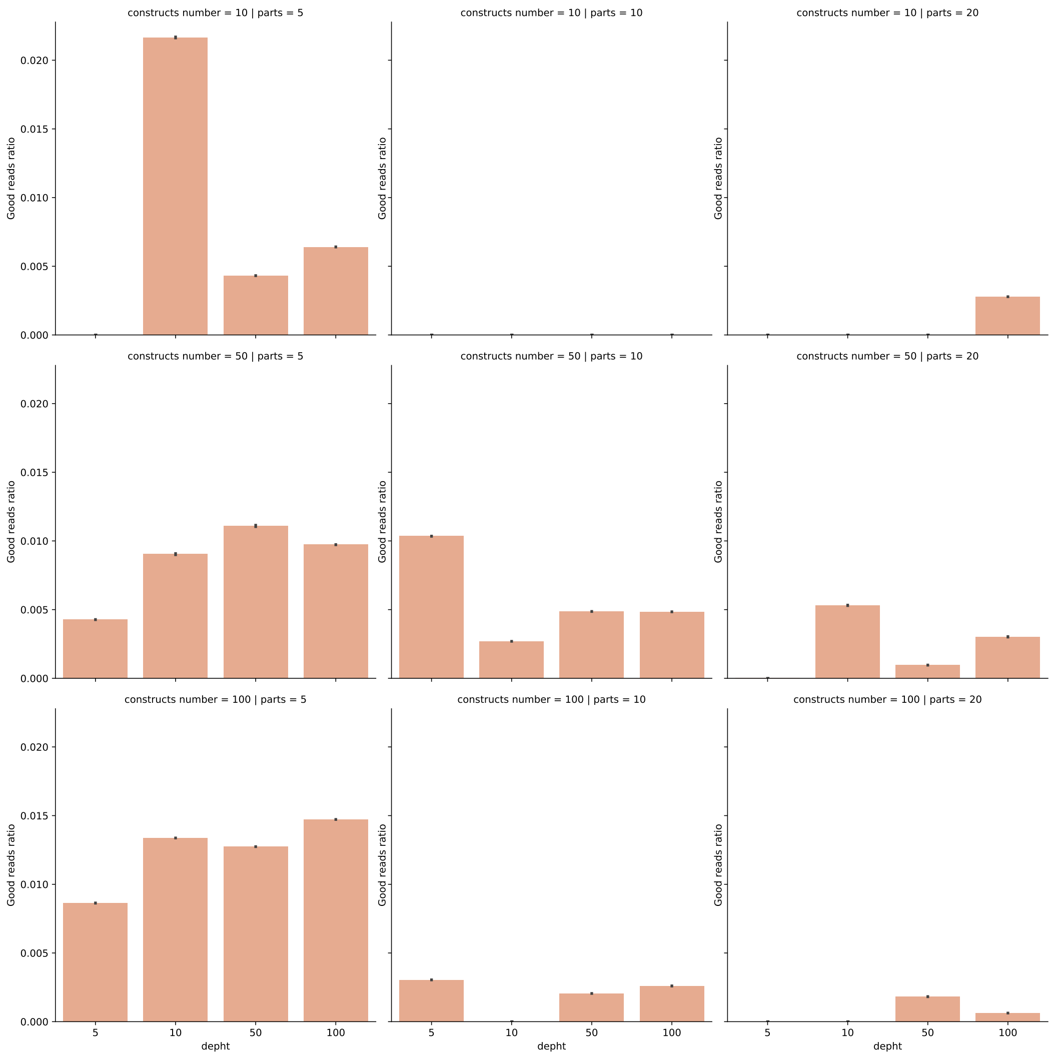

Figure 12: **Good reeads/all reads**

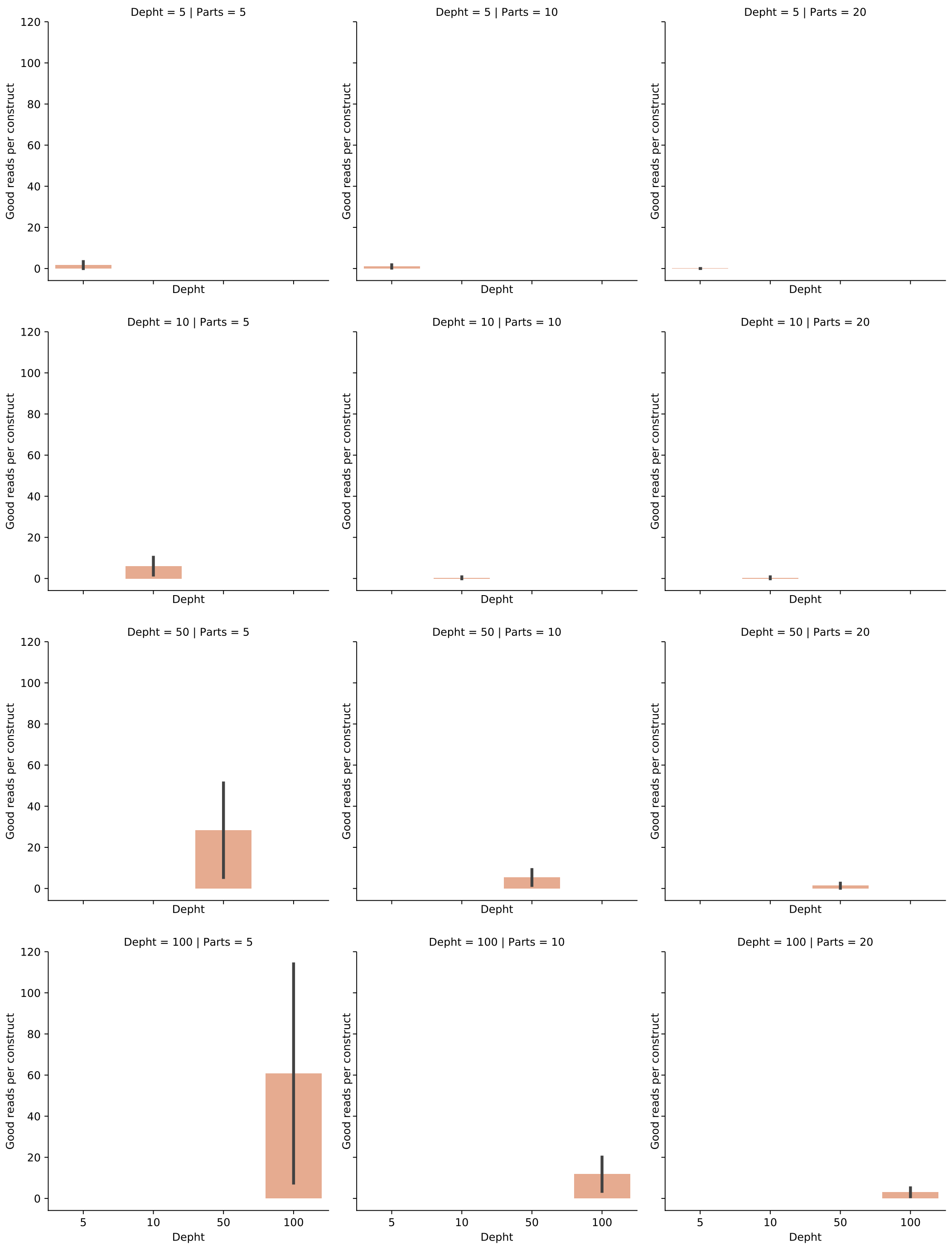

Figure 13: **Good reeads by depth**

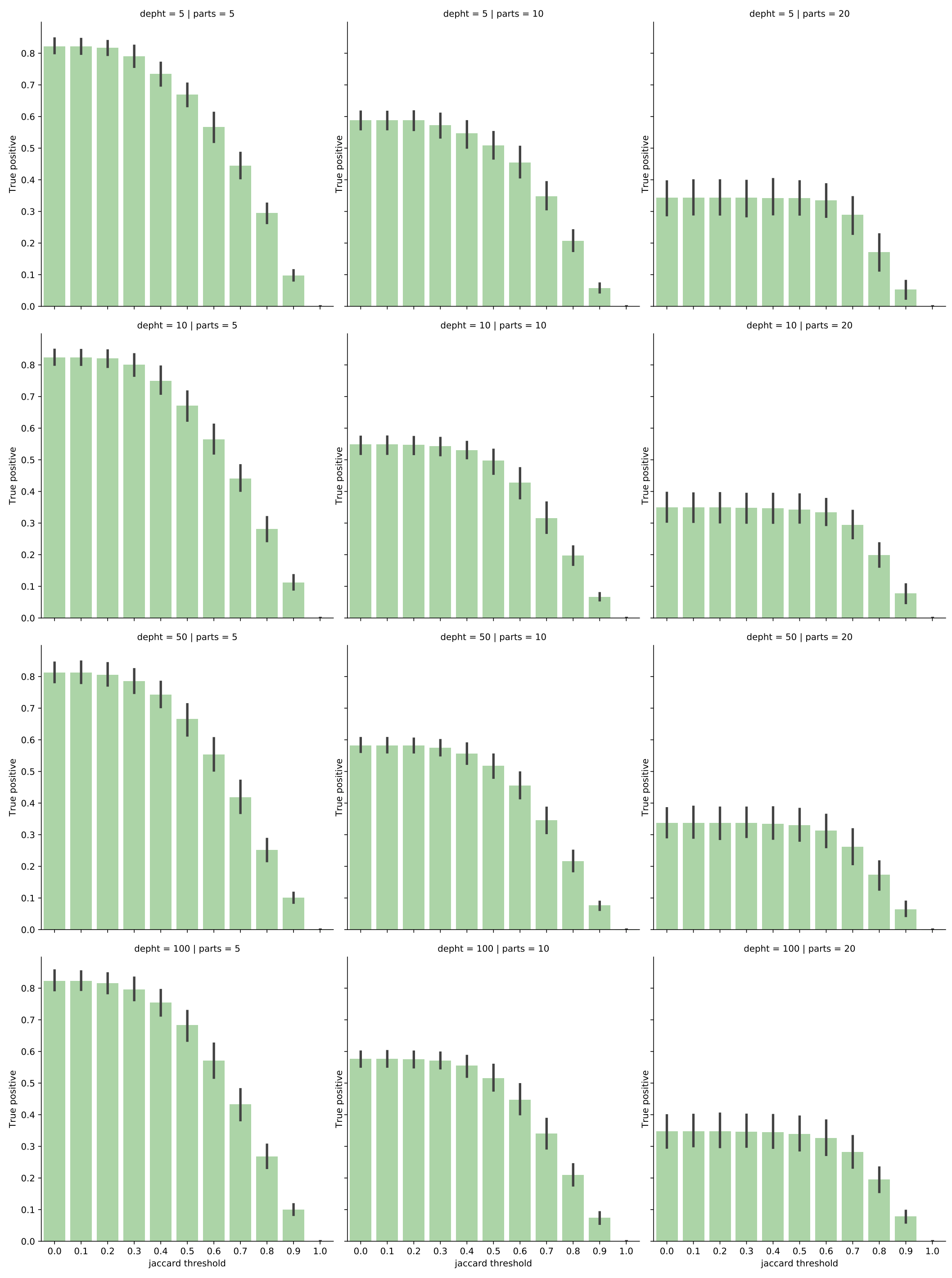

Figure 14: **true postives by depht**

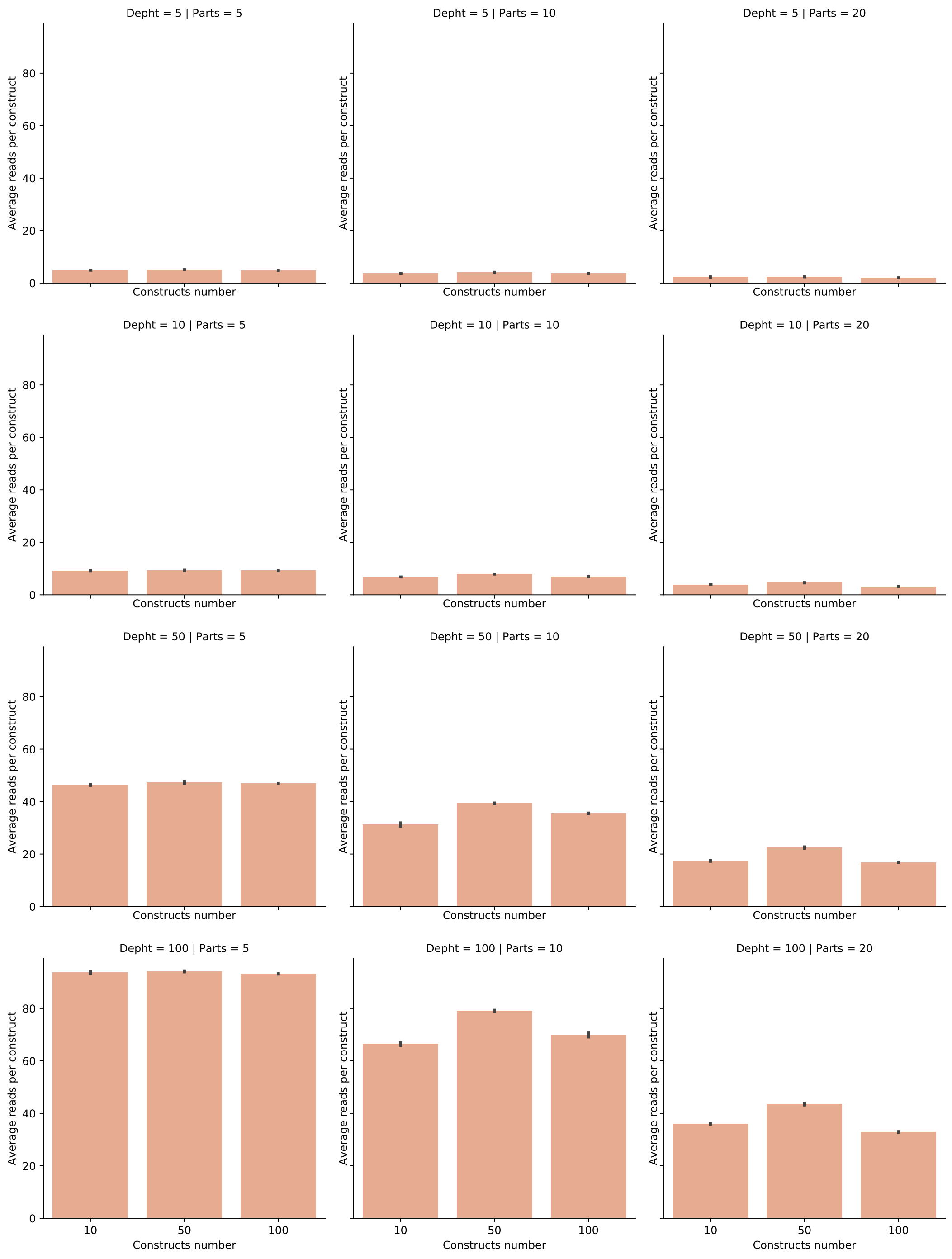

Figure 15: **average reads analysed per construct**

5.3 V3.2.1

5.3.1 Good read filter

Accuracy is defined as: $Goodreadstruepostives/goodreads$ **Good reads** = the number of reads with a length $\geq$ of 99% of the orignal construct **Good reads true positives** = **Good reads** matching with the expected parts in the orignal construct

5.3.2 Good reads amount

Evaluating amount of good reads **Good reads** = the number of reads with a length $\geq$ of 99% of the orignal construct

5.3.3 Good reads ratio

Given by good $reads/allreads$ Evaluating amount of good reads **Good reads** = the number of reads with a length $\geq$ of 99% of the orignal construct

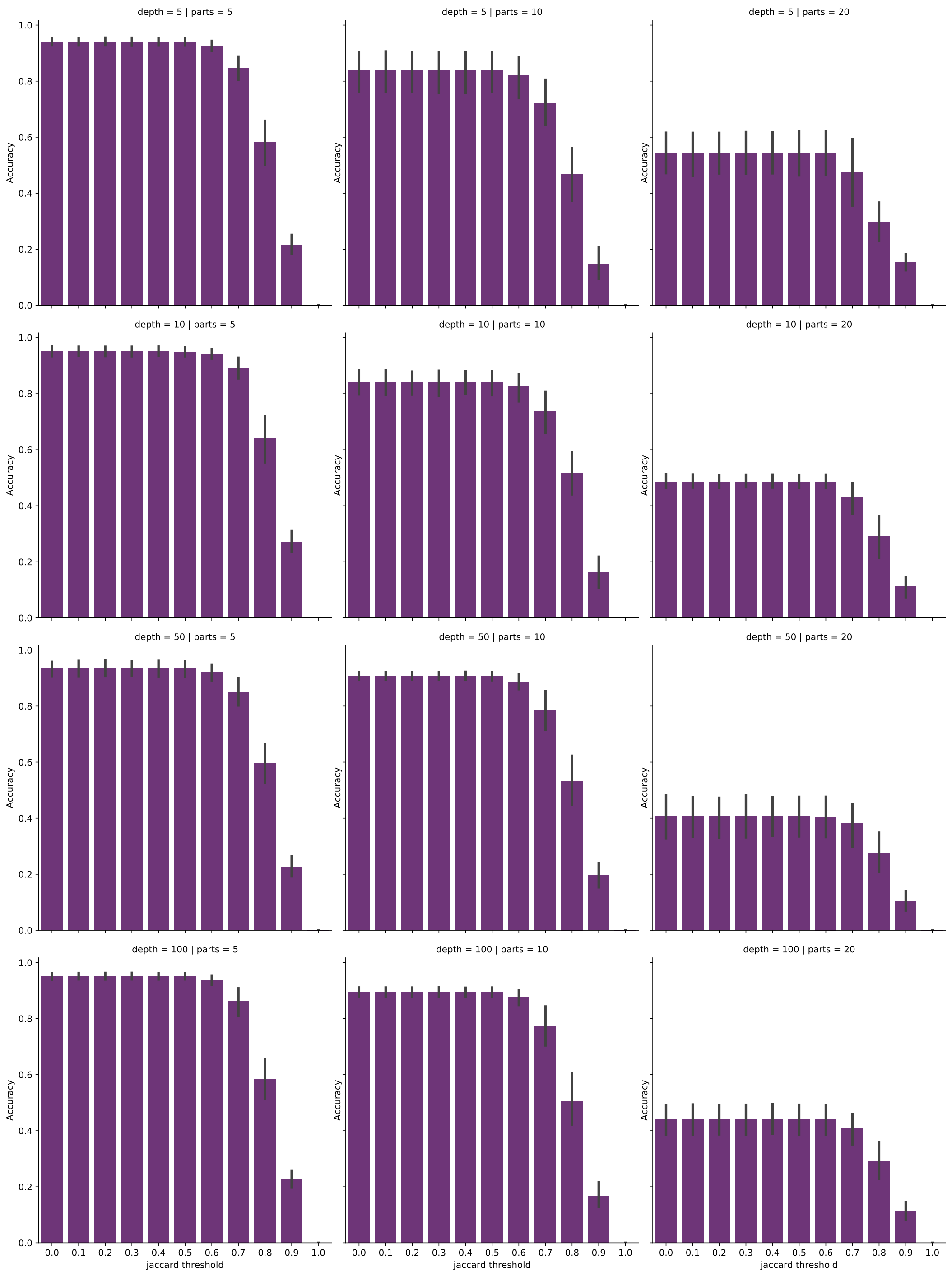

Figure 16: **accuracy** 99%

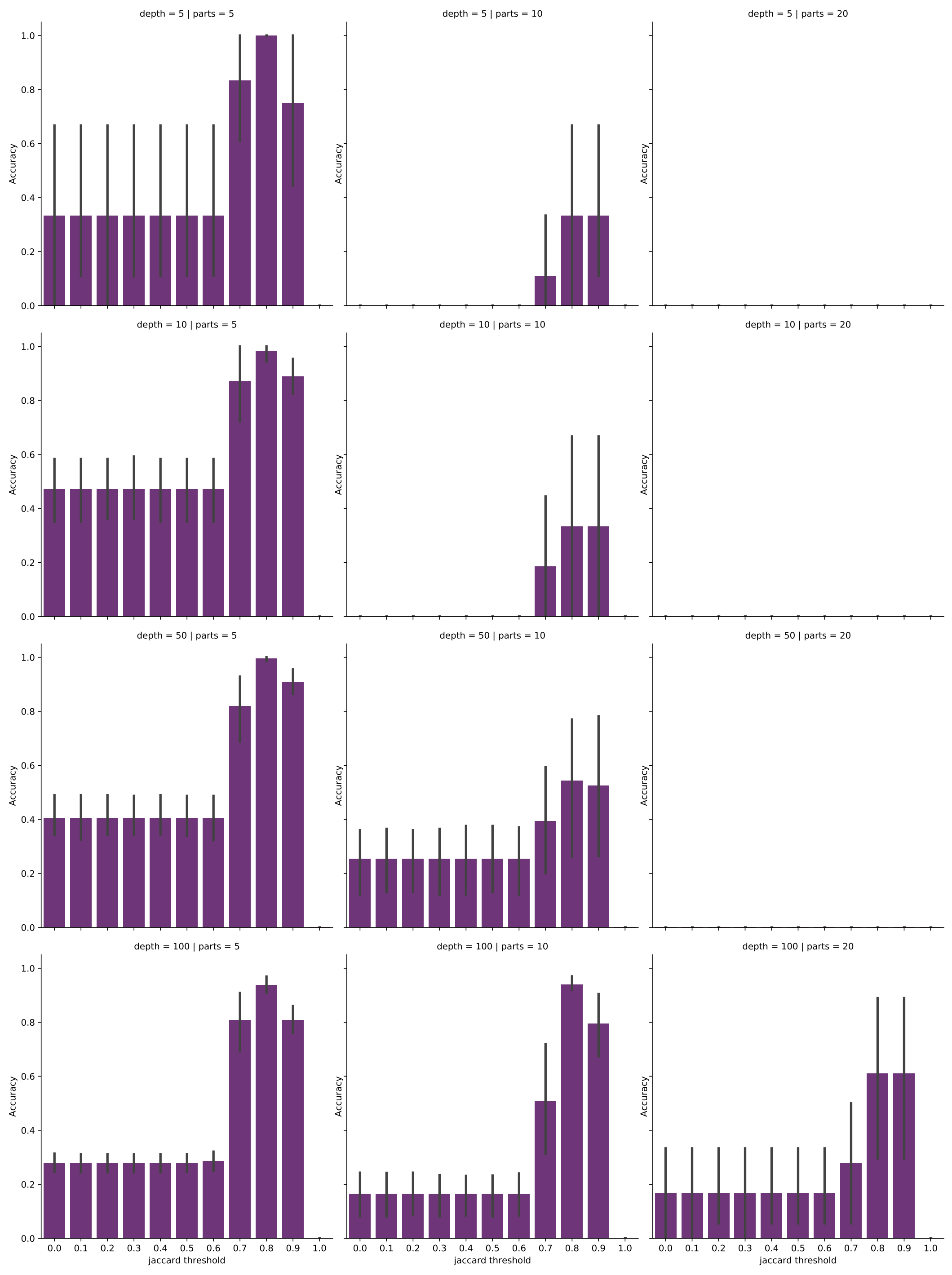

Figure 17: **accuracy** 100%

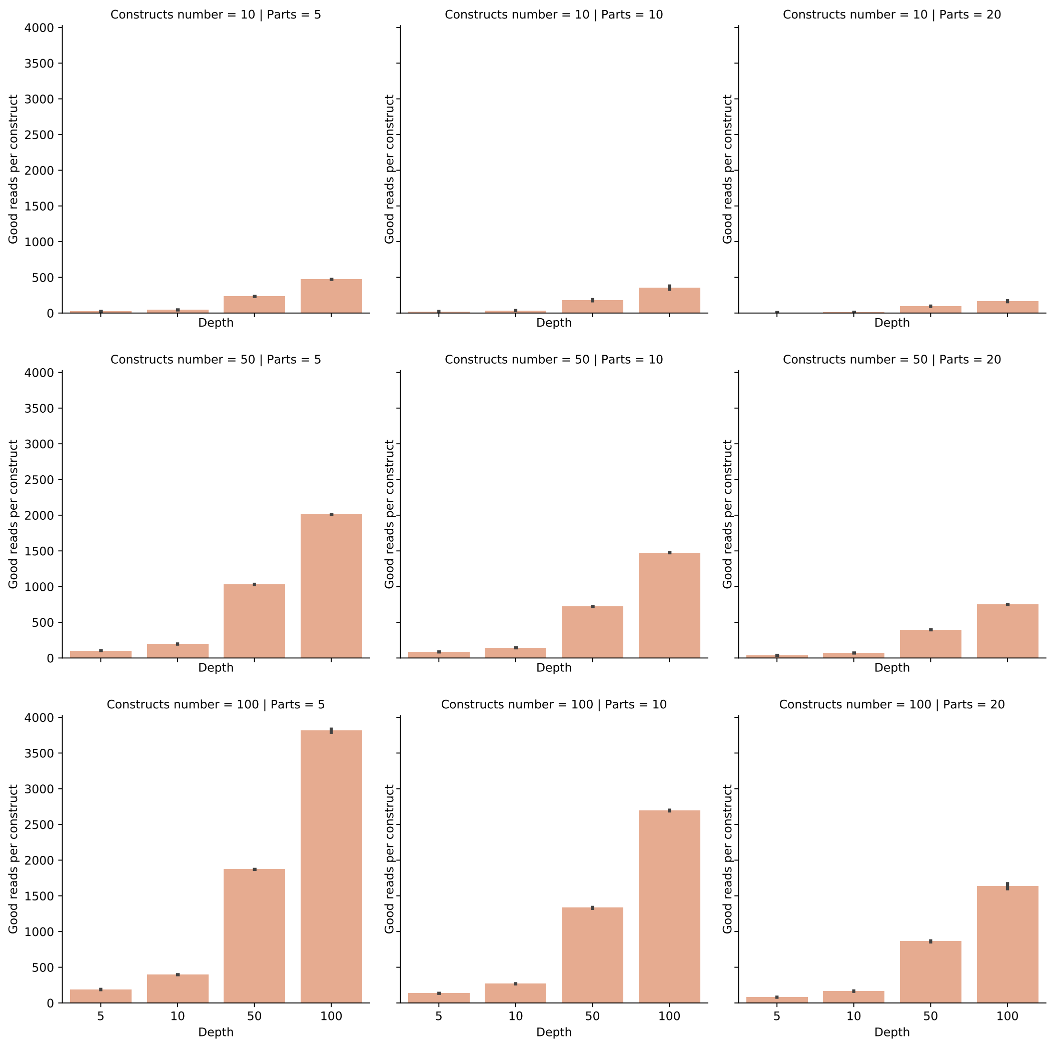

Figure 18: **good reads** 99%

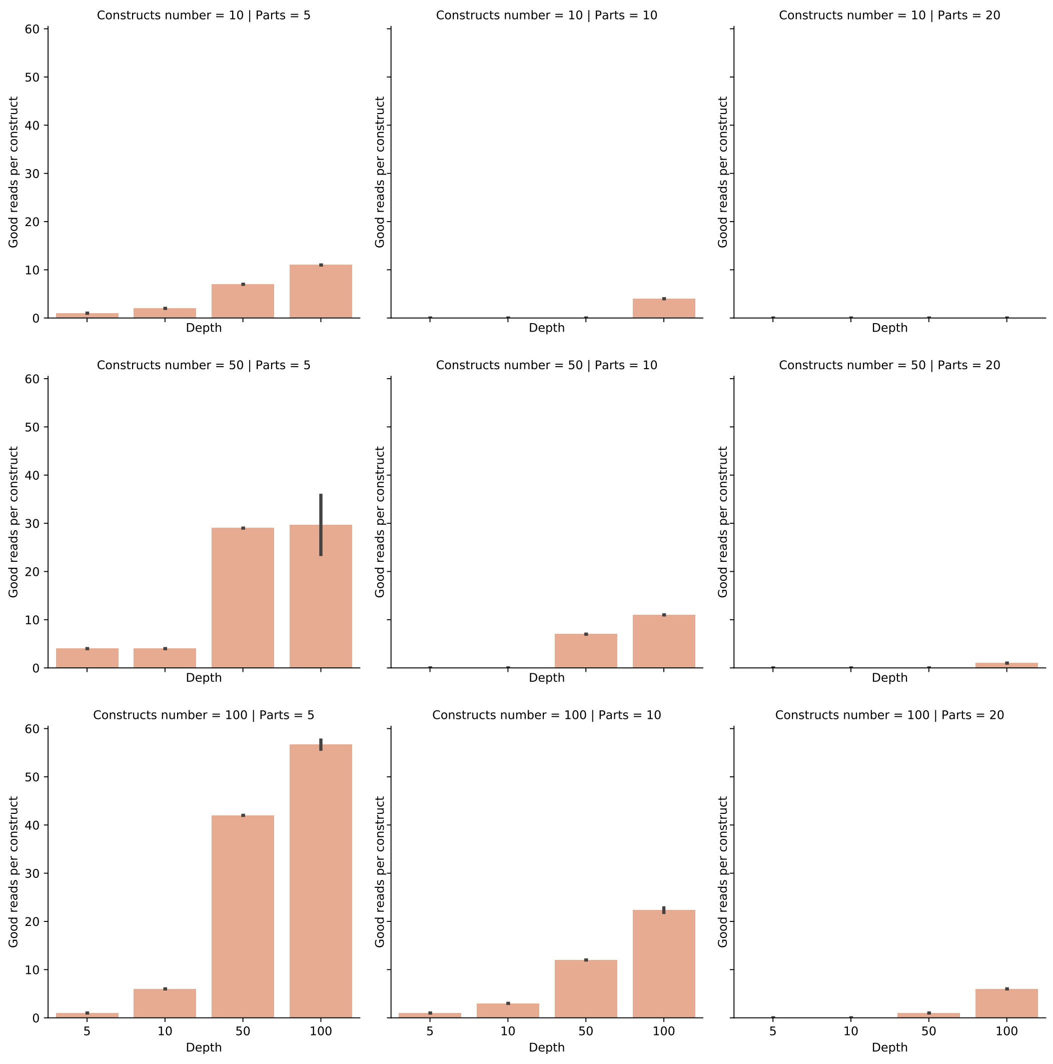

Figure 19: **good reads** 100%

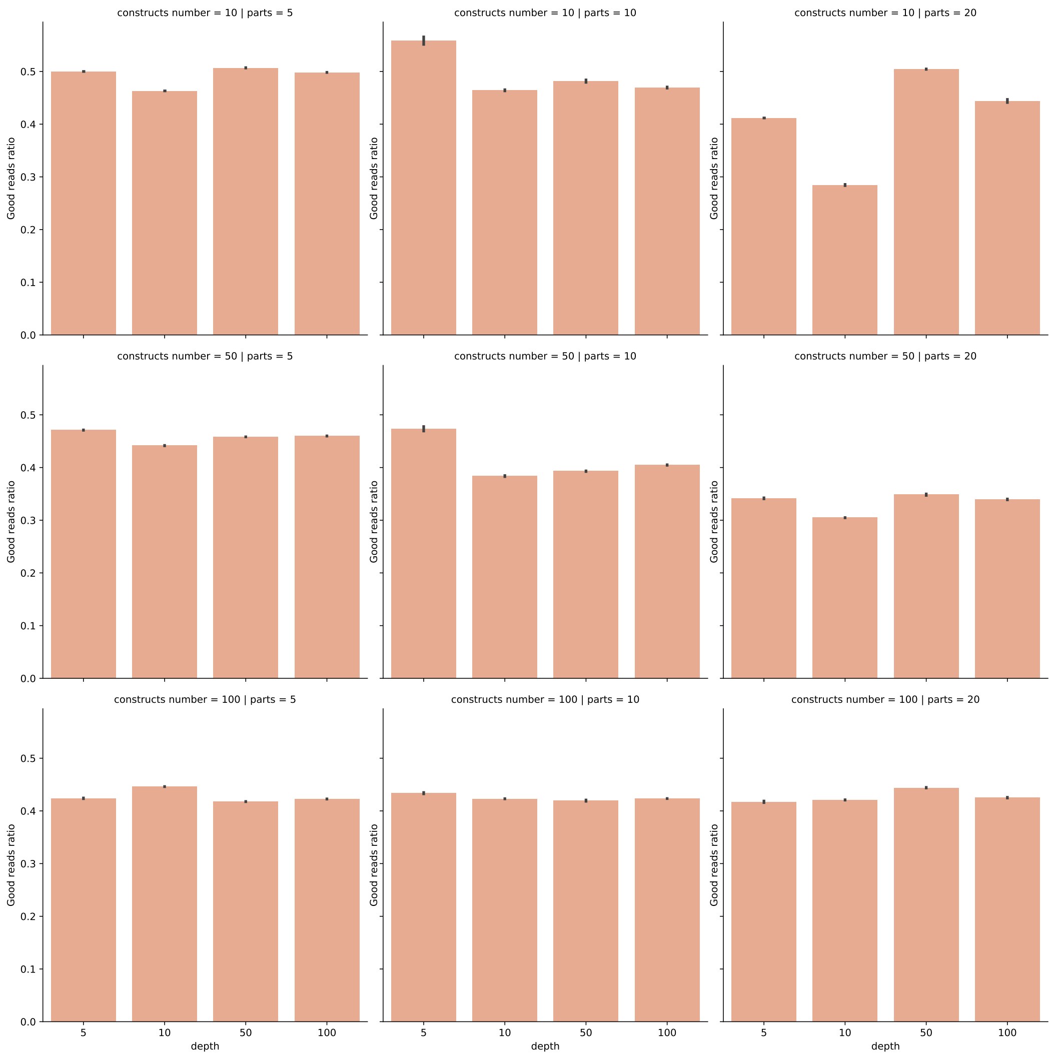

Figure 20: **good reads ratio** 99%

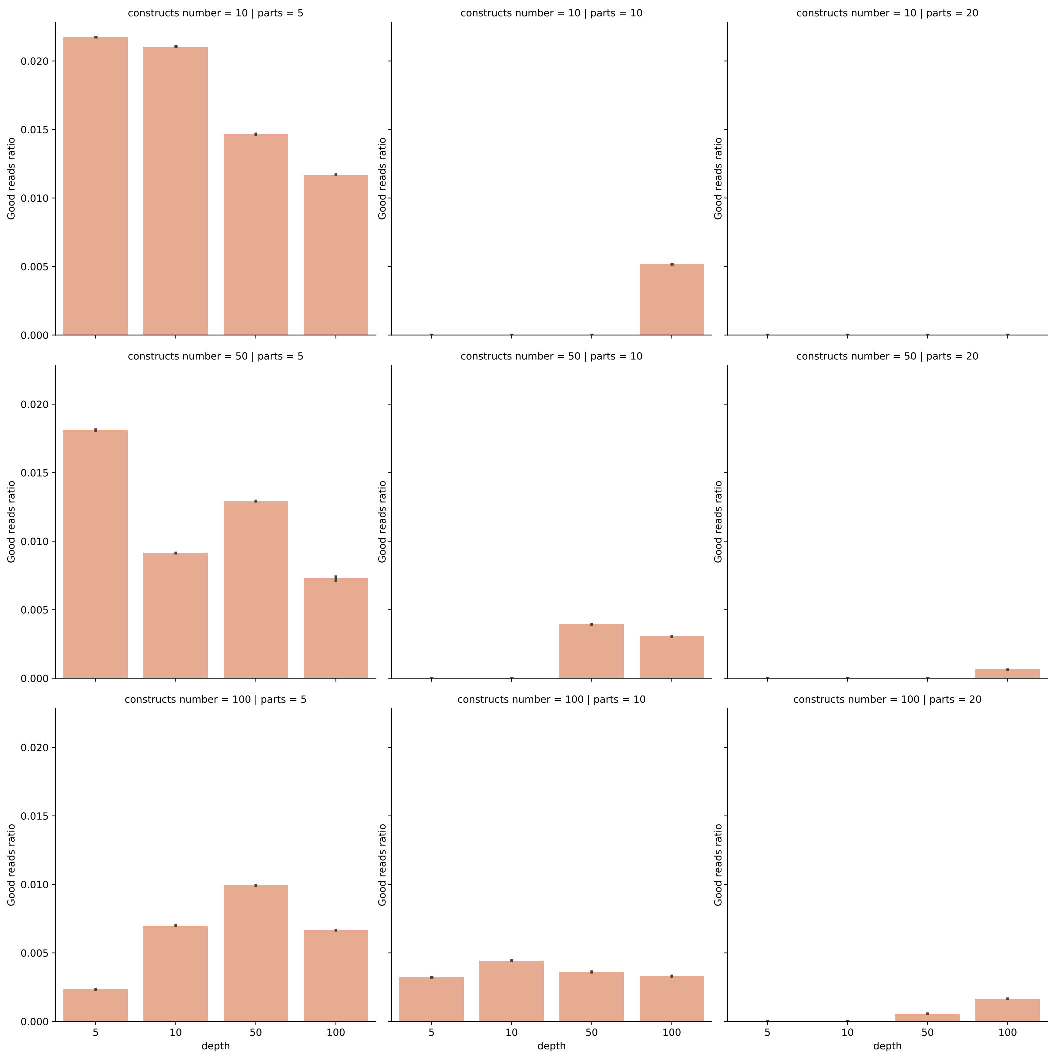

Figure 21: **good reads ratio** 100%

5.4 V3.4.0

Results in this section can be obtained by running the following pipeline:

```
project: "mn_similarity_00"
genbank: "yeast_chromosome_xiv.gb"
run: 10

generate_testbeds:
    cds_number : 50
    constructs_number: [10,50,100]
    parts : [5,10,20]
    constructs_similarity : 0.0
    parts_similarity : 0.0
    mutation_method : "last"

badreads:
    quantity : [5x,10x,50x,100x]
    error_model: "nanopore"
    qscore_model: "nanopore"
    glitches: "0,0,0"
    junk_reads: 0
    random_reads: 0
    chimeras: 0
    identity: 95,100,4
    start_adapter_seq: '␣""␣'
    end_adapter_seq: '␣""␣'

nanogate:
    error_rate : ["001","010","030"]
    jaccard_thresholds: ["00"]
```

5.4.1 nanogate accuracy

Accuracy is defined as: $Goodreadstruepostives/goodreads$ **Good reads** = the number of reads with a length $\geq$ of 99% of the orignal construct **Good reads true positives** = **Good reads** matching with the expected parts in the orignal construct

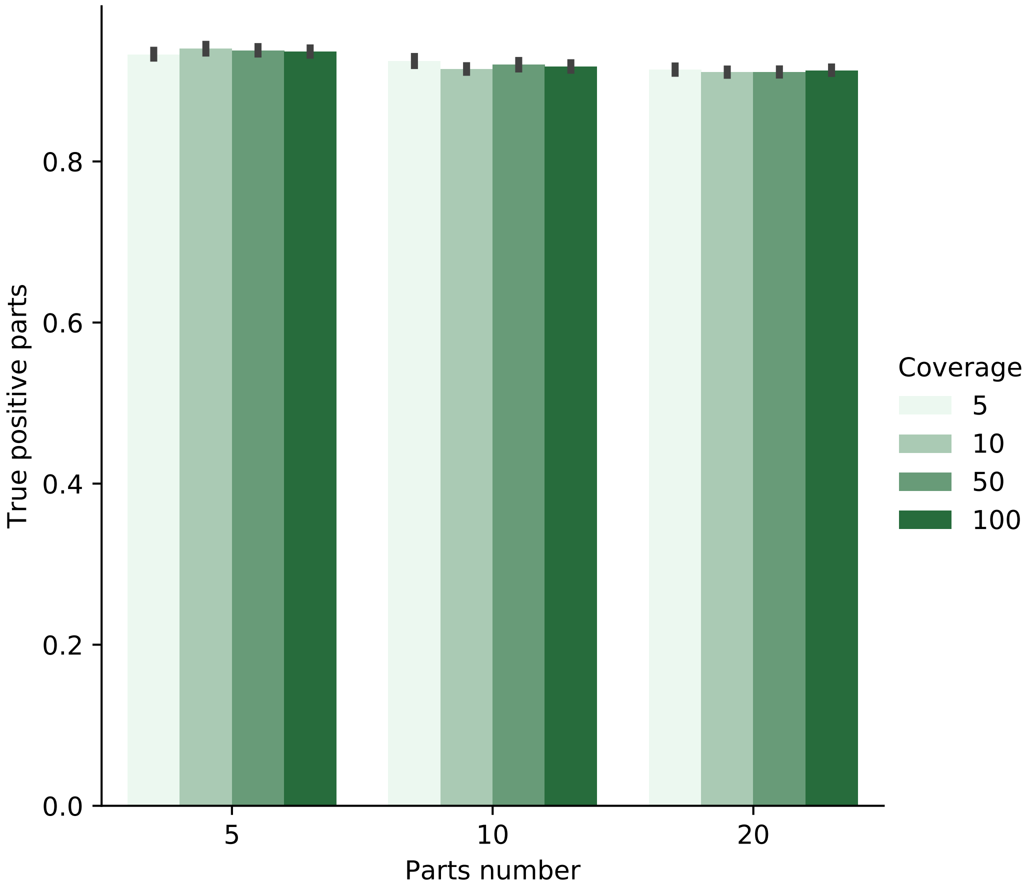

Figure 22: **minimap2 accuracy similarity 0**This plot includes only alignment with alignment alignment score. original reference are aligned to the reads

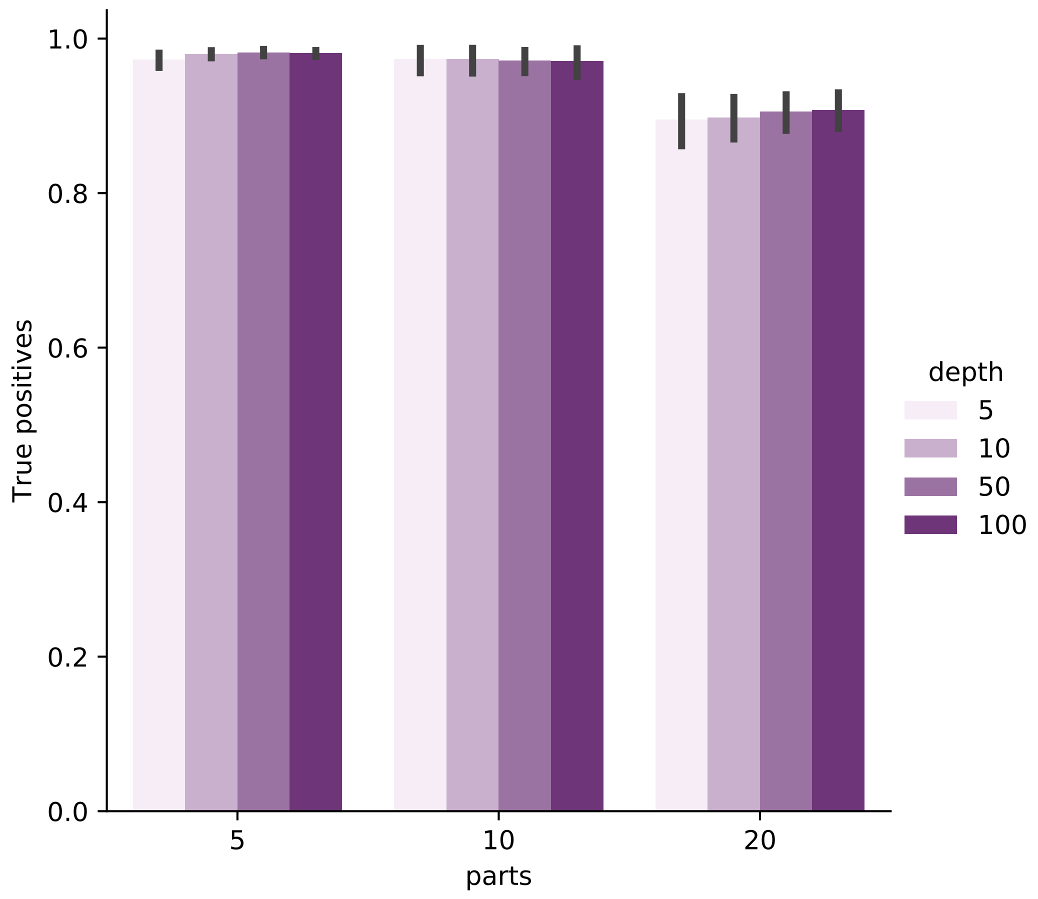

Figure 23: **nanogate accuracy similarity 0** This plot takes into account only reads with a length ¿= 99% of the original construct length

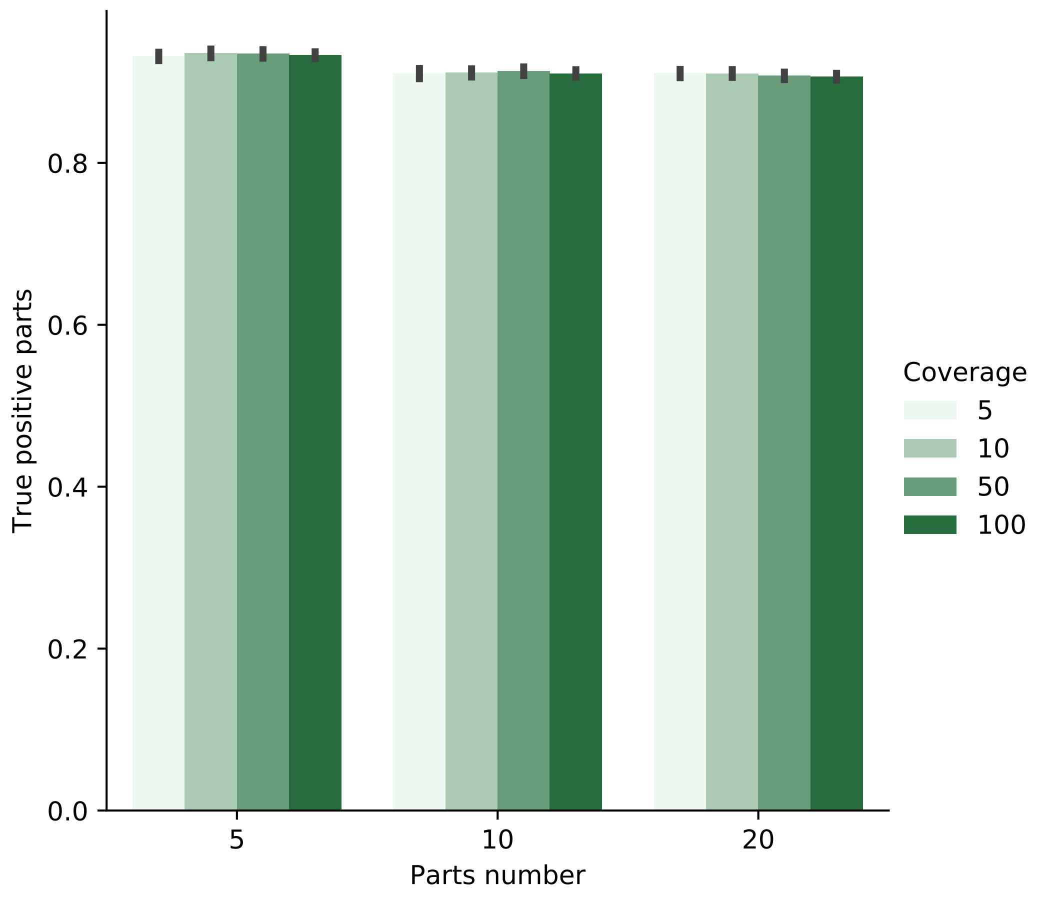

Figure 24: **minimap2 accuracy similarity 70**This plot includes only alligment with alignment alignment score. original reference are aligned to the reads

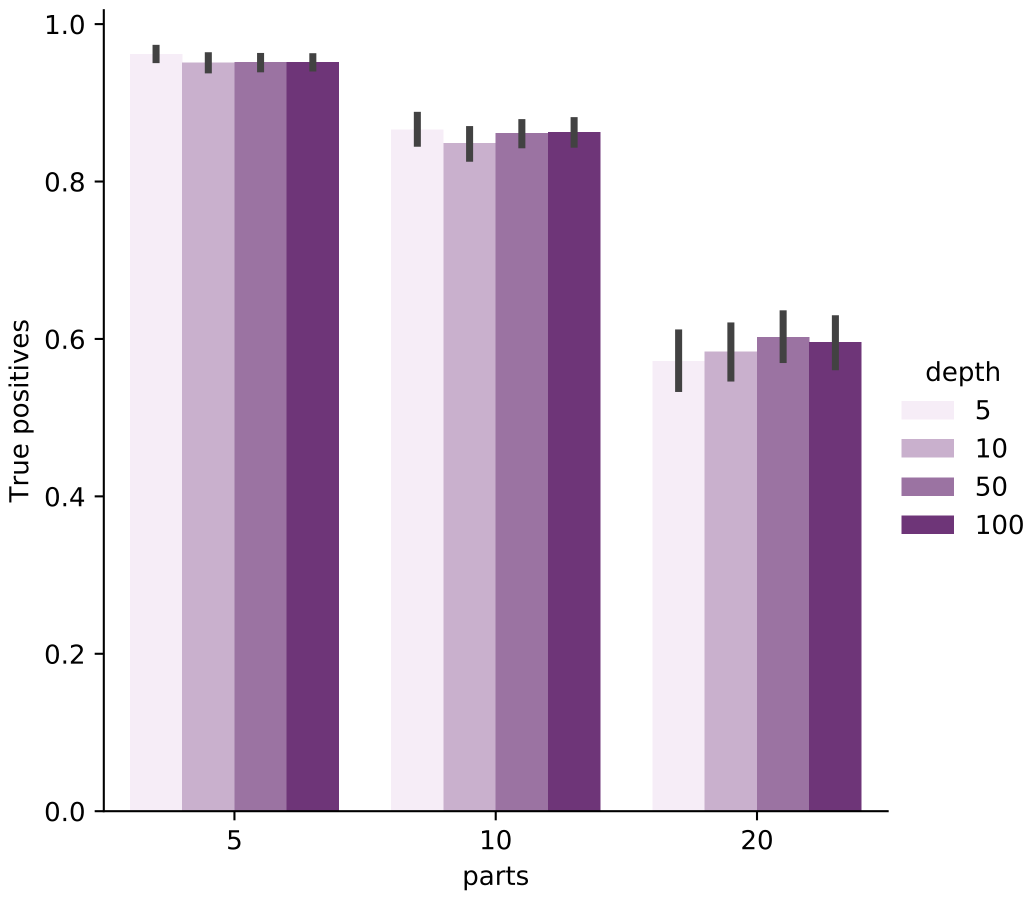

Figure 25: **nanogate accuracy similarity 70** This plot takes into account only reads with a length ¿= 99% of the original construct length

5.5 V4.0.0

Results in this section can be obtained by running the following pipeline:

```
project: "mn_similarity_00"
genbank: "yeast_chromosome_xiv.gb"
run: 10

generate_testbeds:
    cds_number : 30
    constructs_number: [10,50,100]
    parts : [5,10,20]
    constructs_similarity : 0.0
    parts_similarity : 0.0
    mutation_method : "last"

badreads:
    quantity : [5x,10x,50x,100x]
    error_model: "nanopore"
    qscore_model: "nanopore"
    glitches: "0,0,0"
    junk_reads: 0
    random_reads: 0
    chimeras: 0
    identity: 95,100,4
    start_adapter_seq: '␣""␣'
    end_adapter_seq: '␣""␣'

combinatorial_assembly:
    max_parts: 5

nanogate:
     error_rate : ["001","010","030"]
     jaccard_thresholds: ["00"]
```

5.5.1 Minimap2

Plots in this section are related to minimap2

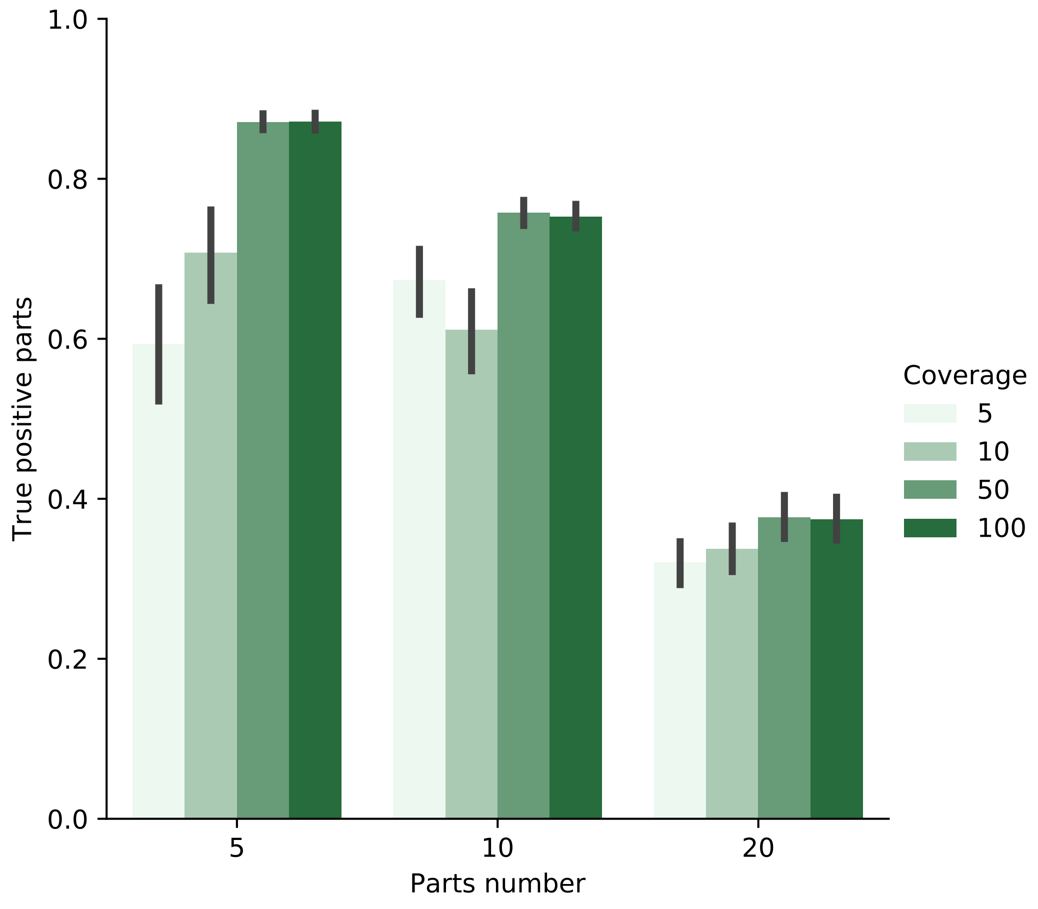

Figure 26: **minimap2 true postives rate without parts similarity.** This plot shows the percentage of reads correctly analysed by Minimap2, unmapped reads and alignments without alignment score are considered as well(combinatorial assembly is not considered here)

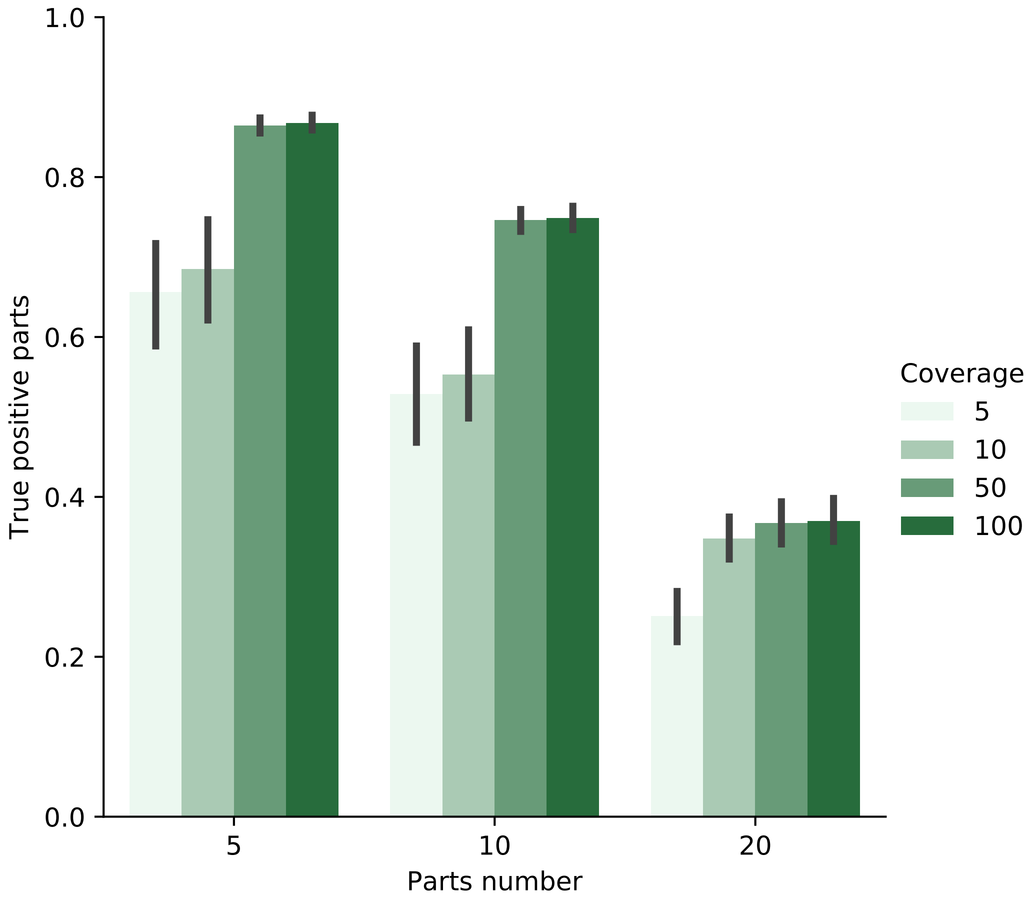

Figure 27: **minimap2 true positives rate with parts similarity = 70%.** This plot shows the percentage of reads correctly analysed by Minimap2, unmapped reads and alignments without alignment score are considered as well(combinatorial assembly is not considered here). Parts similarity slightly affect the results for Minimap2, whereas true positives rate increase with coverage and decrease with the number of parts per construct.

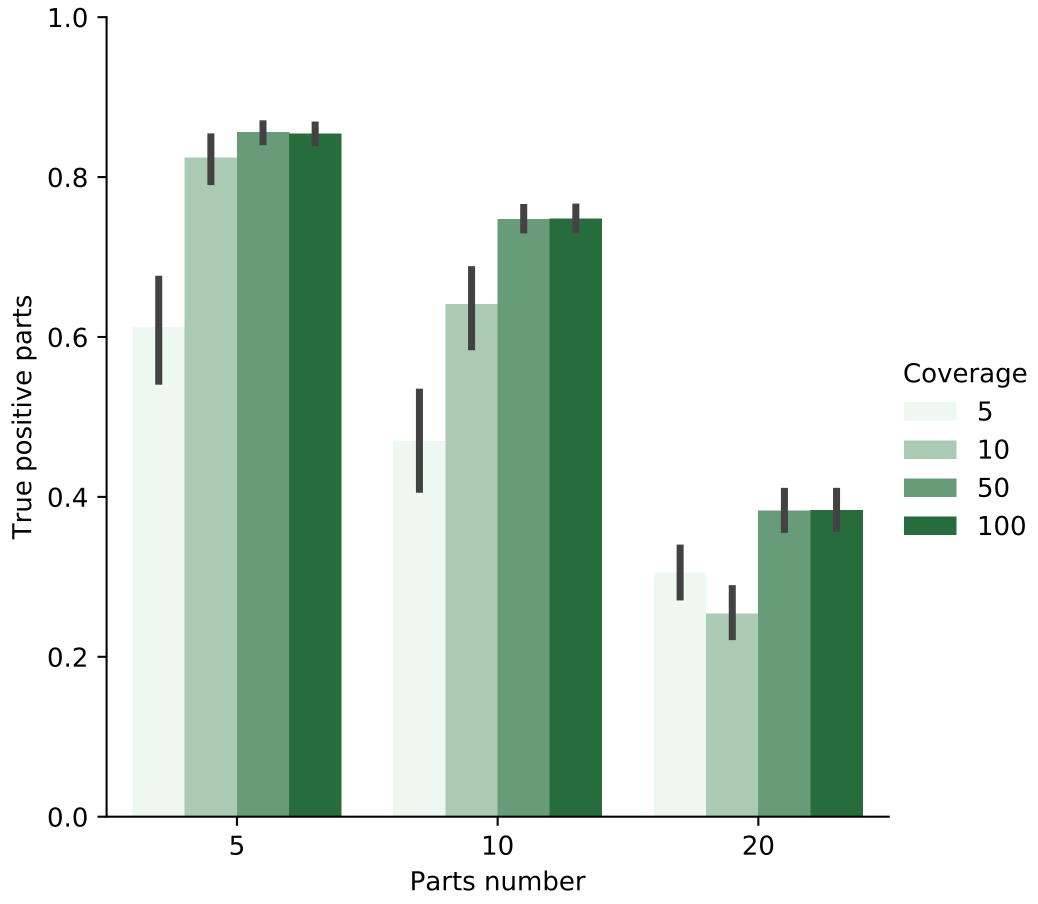

Figure 28: **minimap2 accuracy with parts similarity = 90%.** This plot shows the percentage of reads correctly analysed by Minimap2, unmapped reads and alignments without alignment score are considered as well(combinatorial assembly is not considered here). Parts similarity slightly affect the results for Minimap2, whereas true positives rate increase with coverage and decrease with the number of parts per construct.

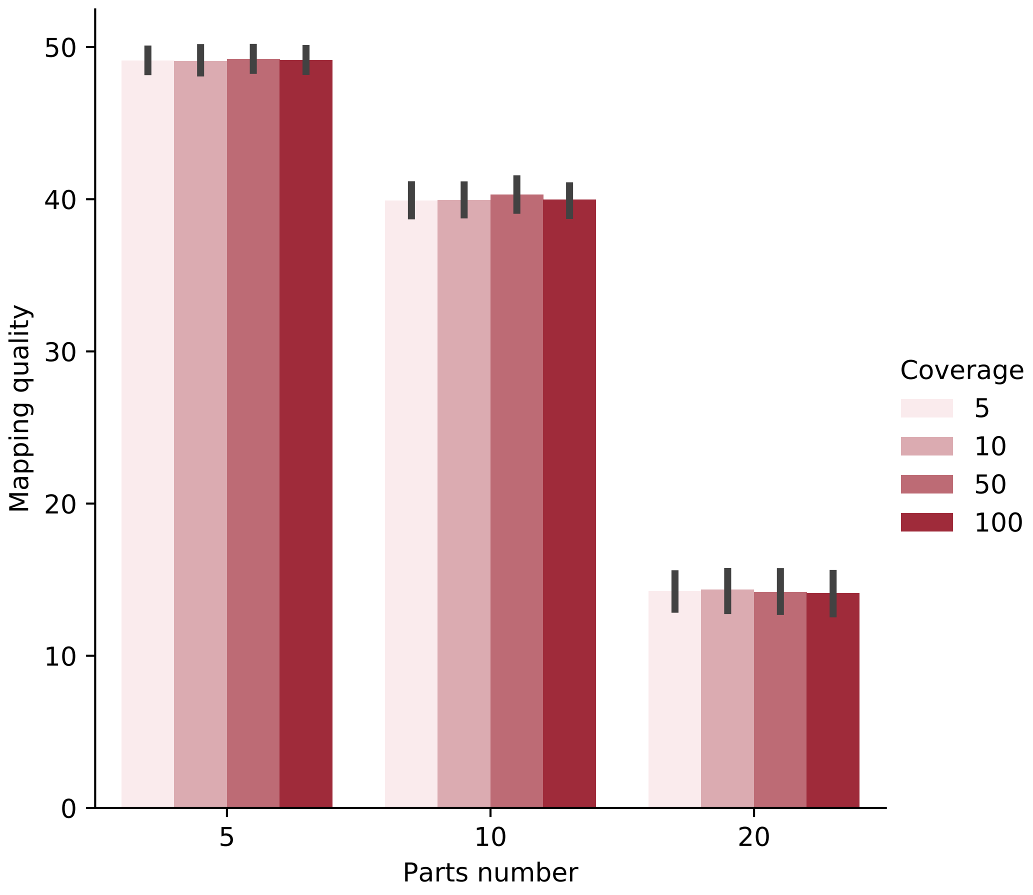

Figure 29: **minimap2 mapping quality without parts similarity.** This plot shows how the Minimap2 mapping quality vary with coverage and number of parts per construct (combinatorial assembly is not considered here).Coverage seem to not affect mapping quality, however it decrease with the number of parts per construct

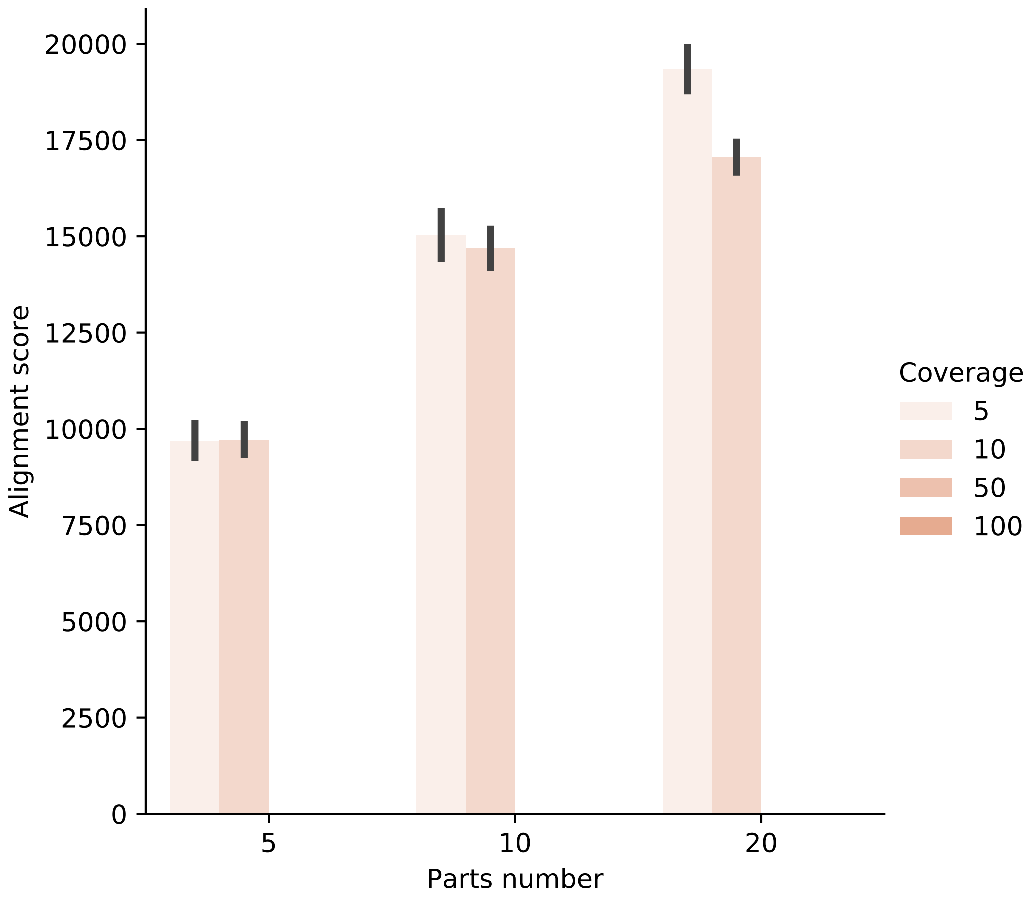

Figure 30: **minimap2 alignment score without parts similarity.** This plot shows how the Minimap2 alignment score vary with coverage and number of parts per construct (combinatorial assembly is not considered here).The alignment score increase with the number of parts per constructs, since as the length of a construct increase the number of matching bases increase as well. Coverage 50 and 100 are not shown need further investigation

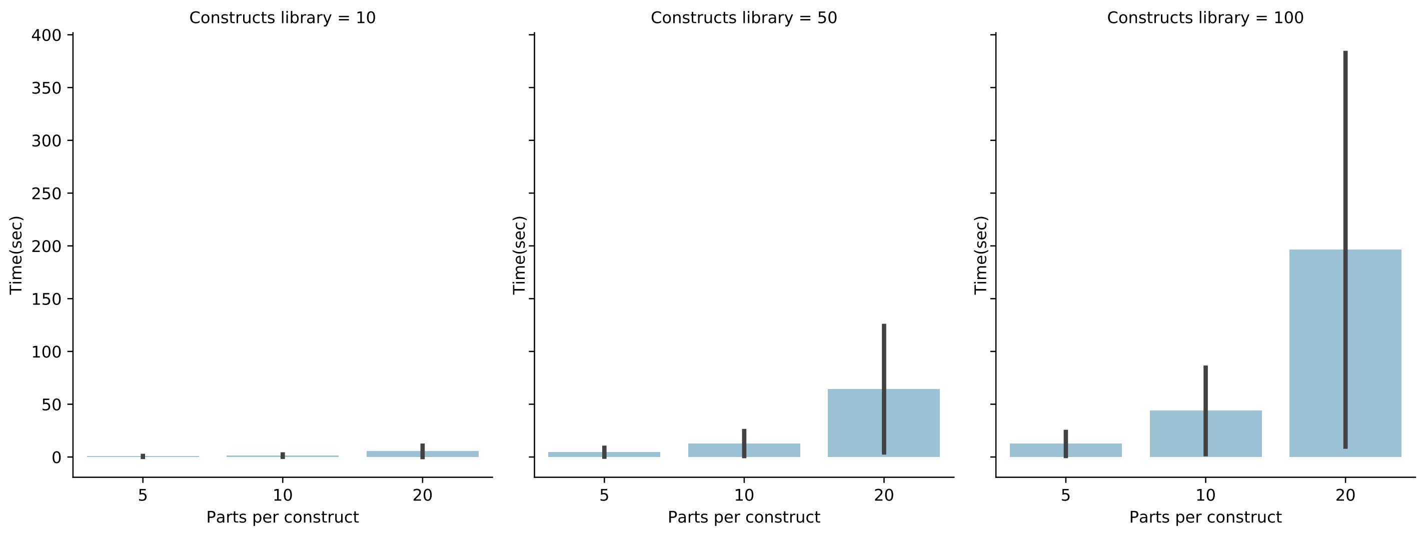

Figure 31: **minimap2 Real time without parts similarity.** This plot shows the time required by Minimap2 to per perform alignments between constructs generated by testbed generator and the reads generated by badreads tool, (combinatorial assembly is not considered here)

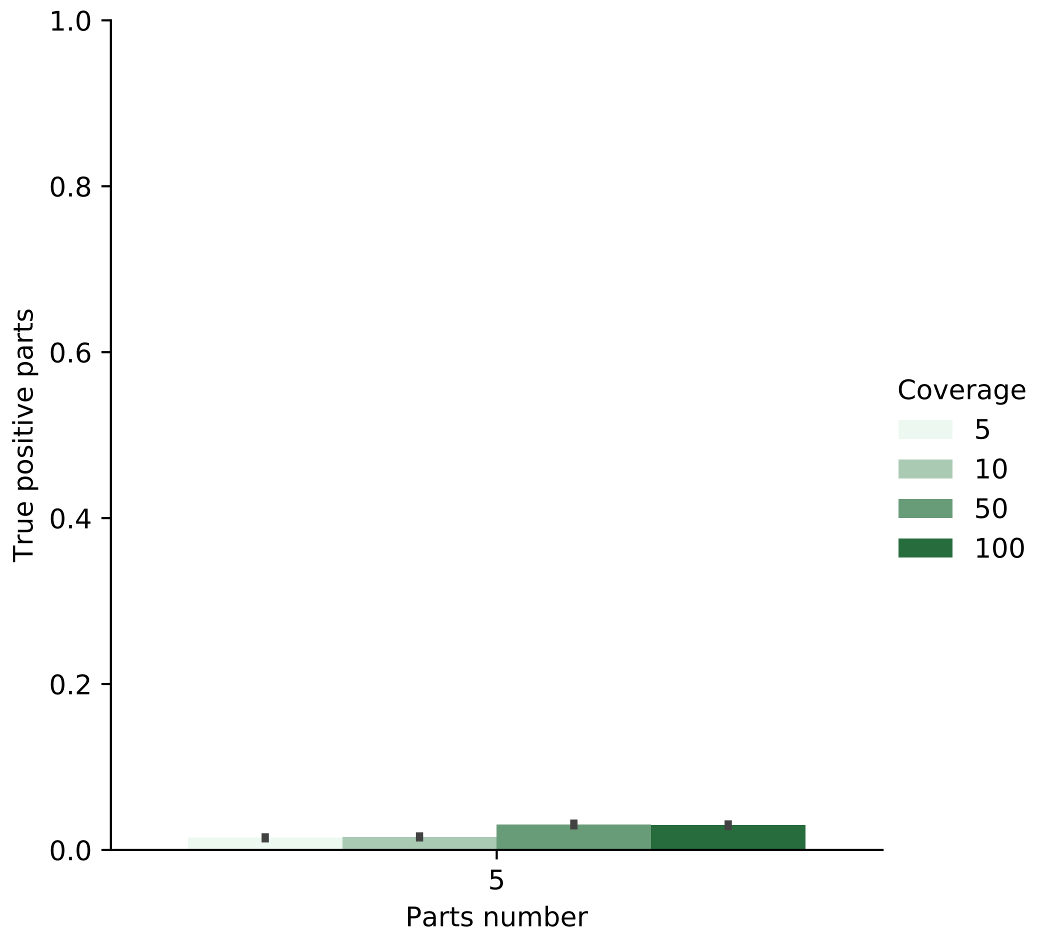

Figure 32: **minimap2 combinatorial assembly true positives rate without parts similarity.** This plot shows the true positives rate obtained after performing an alignment between reads generated from badreads tool and parts assembled combinatorially. To save time on the analysis only combinatorial assembly for 5 parts constructs are evaluated. The true positives rate slightly increase with coverage, however is always below 0.1.

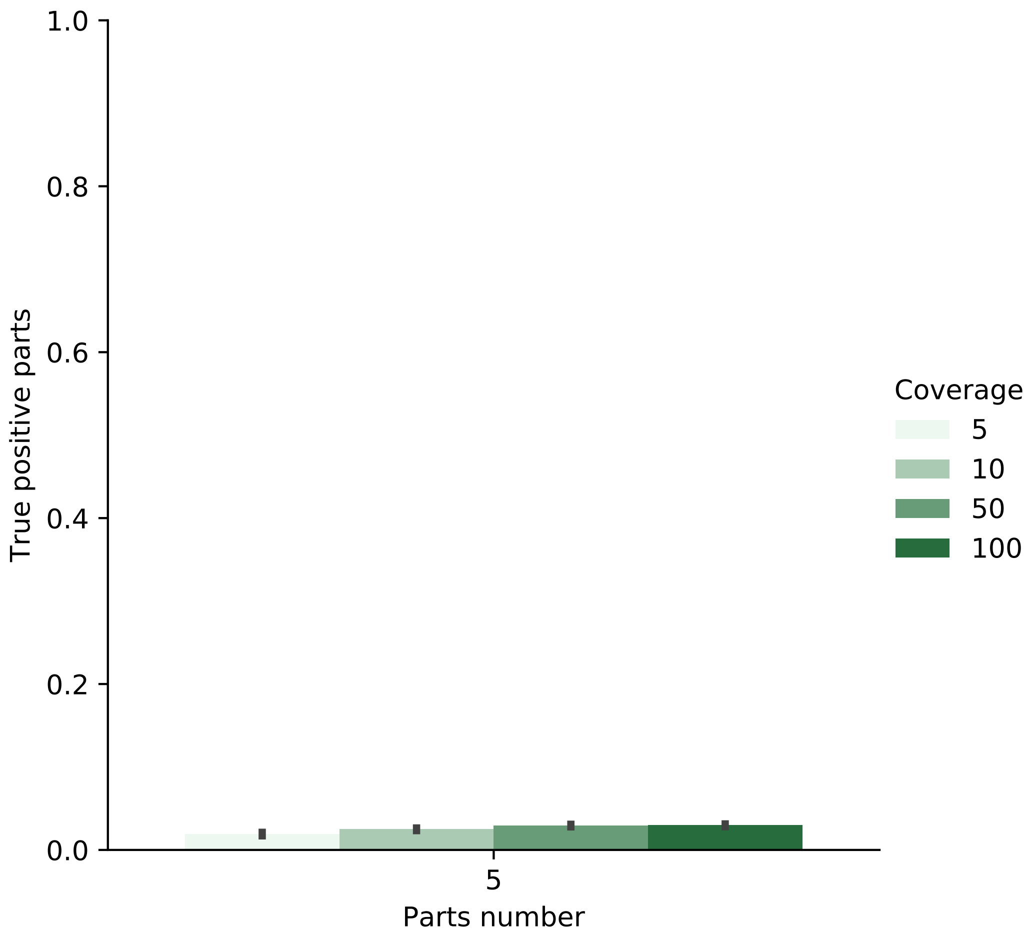

Figure 33: **minimap2 combinatorial assembly true positives rate with parts similarity =90%.** This plot shows the true positives rate obtained after performing an alignment between reads generated from badreads tool and parts assembled combinatorially. To save time on the analysis only combinatorial assembly for 5 parts constructs are evaluated. The true positives rate slightly increase with coverage, however is always below 0.1.

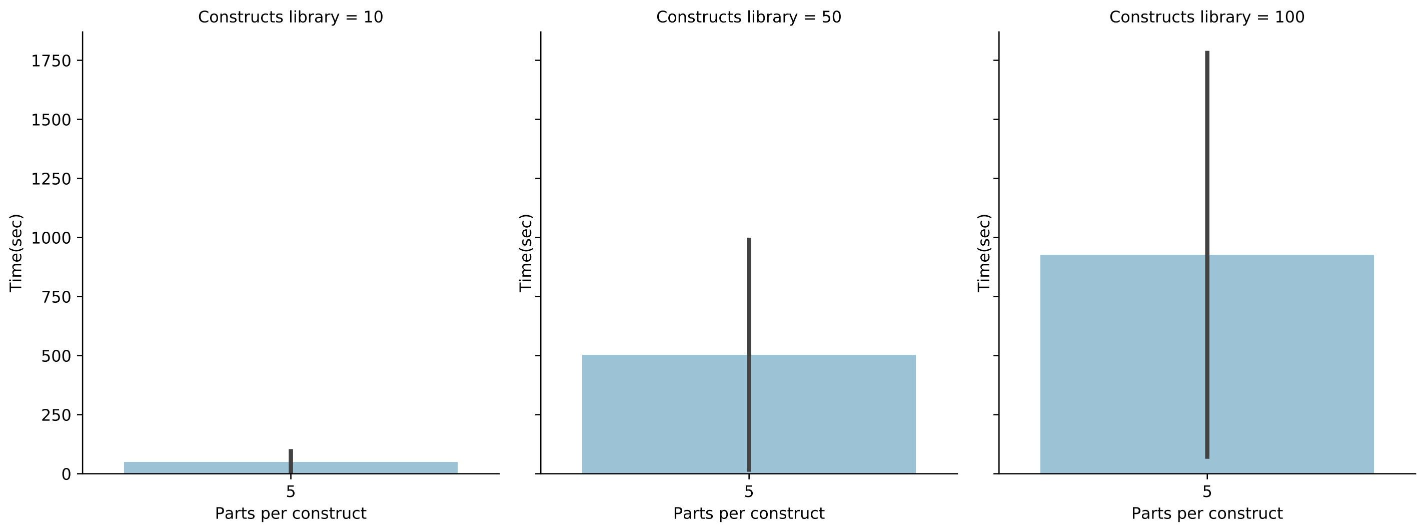

Figure 34: **minimap2 combinatorial assembly real time wihtout parts similarity.** This plot shows the time required by minimap2 to perform an alignment between parts combinatorially assembled used as reference and reads generated by badreas tool as query

5.5.2 Nanogate

Plots in this section are related to Nanogate

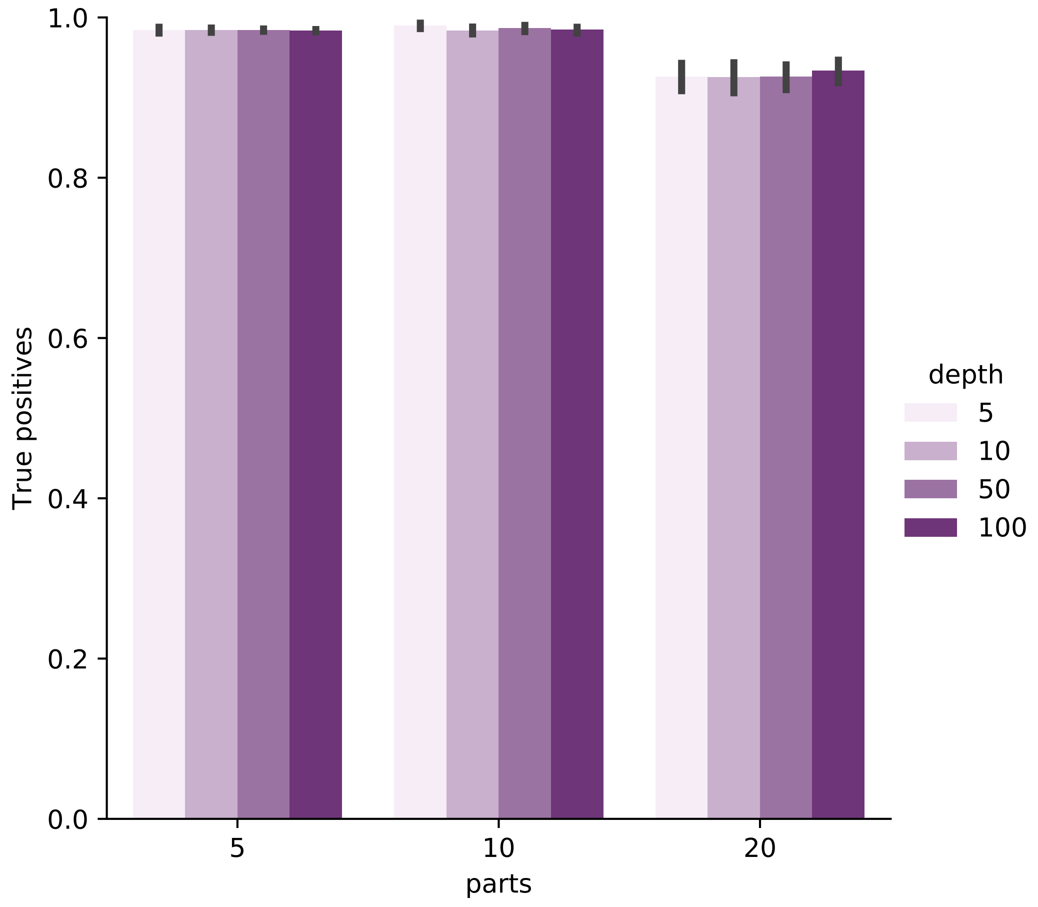

Figure 35: **Nanogate true postives rate without parts similarity.** This plot shows the percentage of reads correctly analysed by Nanogate, only reads with a length $>= 99\%$ than the constructs original length are considered here. With no parts similarity Nanogate gives a true positives rate close to 1, and give better result compared to Minimap2 whether in standard alignments or applied to combinatorial assembly.

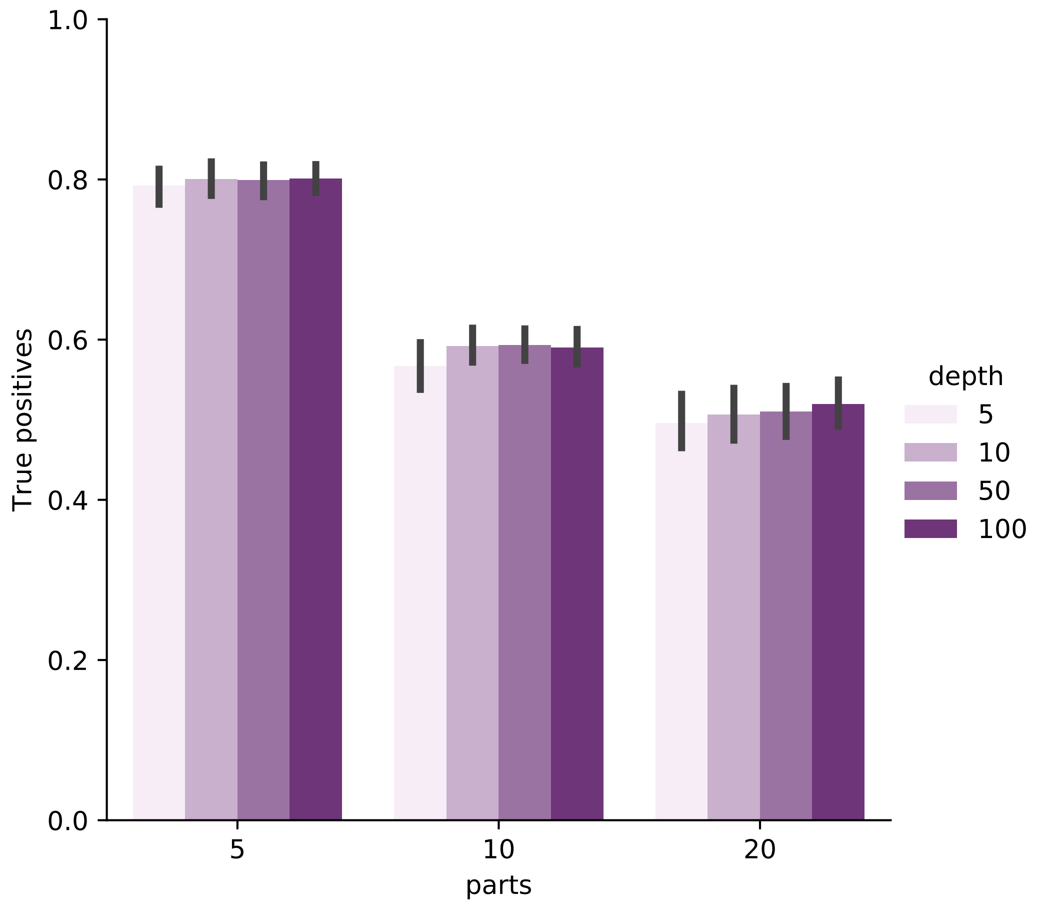

Figure 36: **Nanogate true positives rate with parts similarity =90%.** This plot shows the percentage of reads correctly analysed by Nanogate, only reads with a length $>= 99\%$ than the constructs original length are considered here. With no parts similarity Nanogate gives a true positives rate close to 1, and give better result compared to Minimap2 whether in standard alignments or applied to combinatorial assembly. However, compared to analysis performed without parts similarity the true positives rate has sensibly decreased in particular for 10 and 20 parts constructs

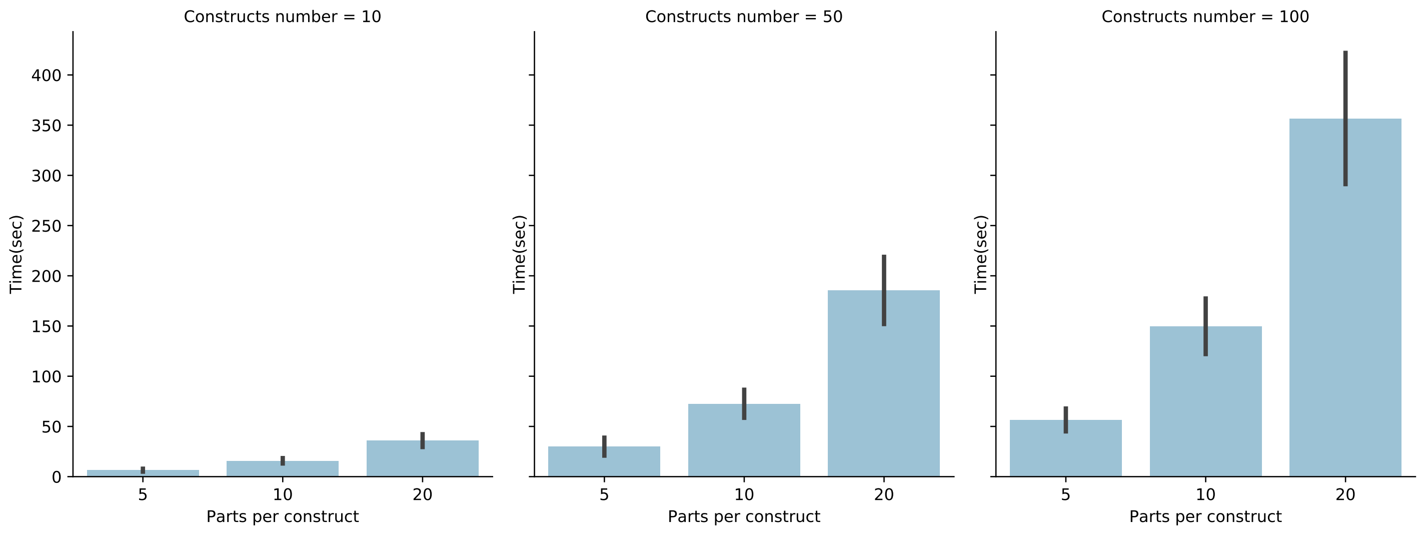

Figure 37: **Nanogate time plot.** This plot shows the time required by Nanogate to perform the analysis, on average if compared to Minimap2 applied in standard condition, Nanogate require on average more time to perform the analysis. If we compare the computational time bewteen Nanogate and Minimap2 applied to combinatorial assembly, Nanogate requires by far less time.

5.5.3 Minimap2 Nanogate Comparisons

Plots in this section show comparisons between Minimpa2 and Nanogate

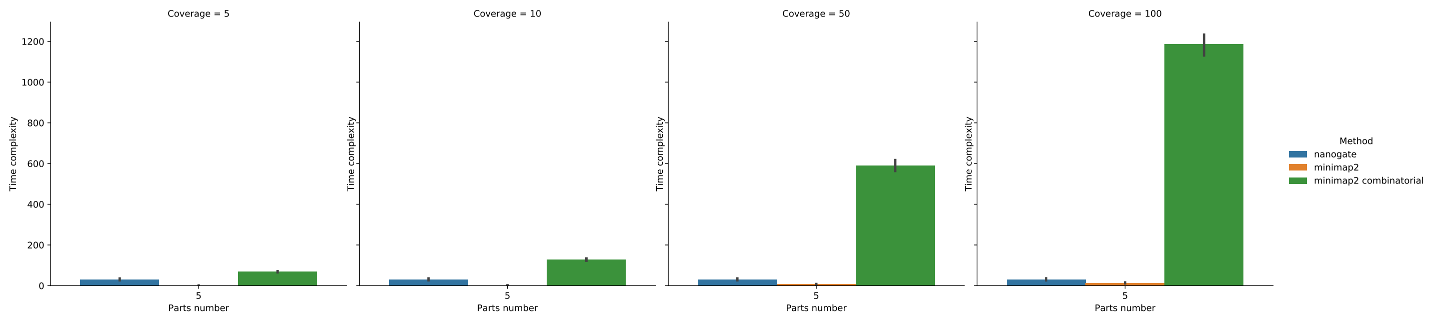

Figure 38: **Nanogate Minimap2 and combinatorial assembly time plot.** This plot shows, in absence of parts similarity, the time complexity comparison between Nanogate, Minimap2 using as query only constructs reads, and Minimap2 applied to combinatorial assemblies. To simplify the analysis only 5 parts constructs are taken into account.

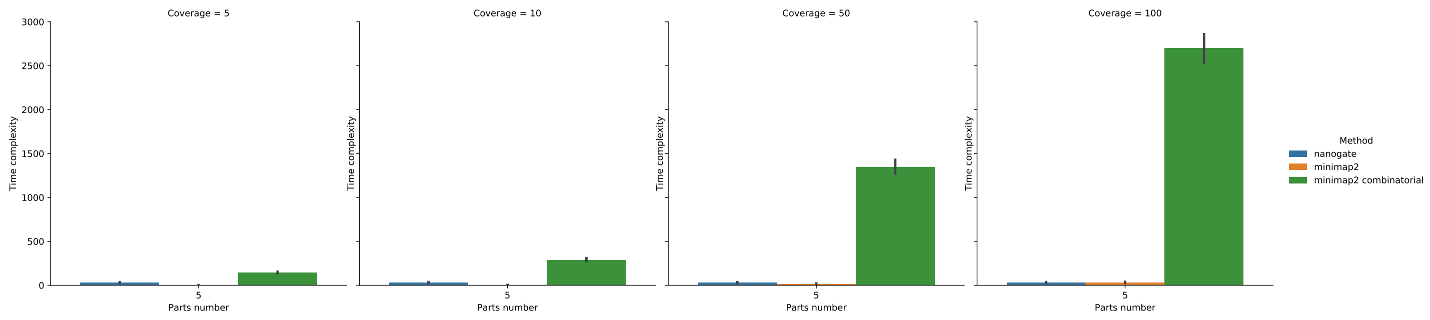

Figure 39: **Nanogate Minimap2 and combinatorial assembly time plot.** This plot shows, for parts similarity $= 90\%$, the time complexity comparison between Nanogate, Minimap2 using as query only constructs reads, and Minimap2 applied to combinatorial assemblies. To simplify the analysis only 5 parts constructs are taken into account.

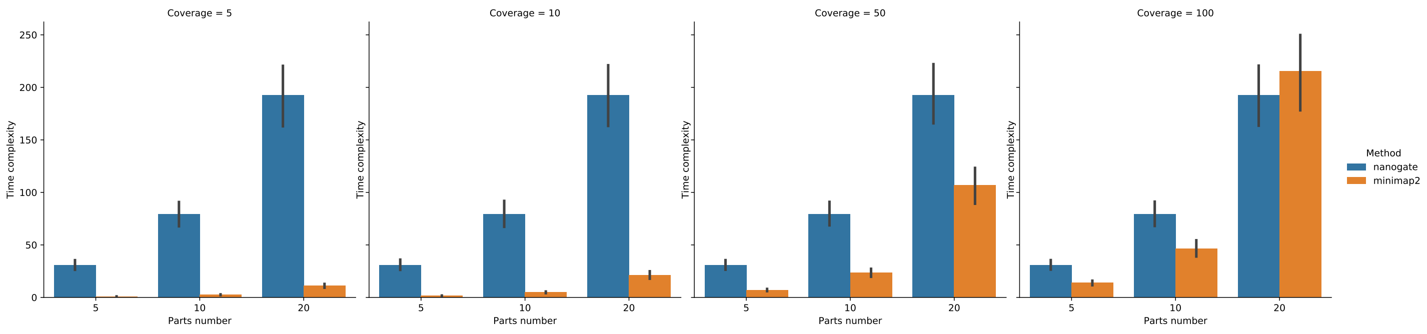

Figure 40: **Nanogate and Minimap2 time comparison plot.** This plot shows, in absence of parts similarity, the time complexity comparison between Nanogate and Minimap2 using as query only constructs reads.

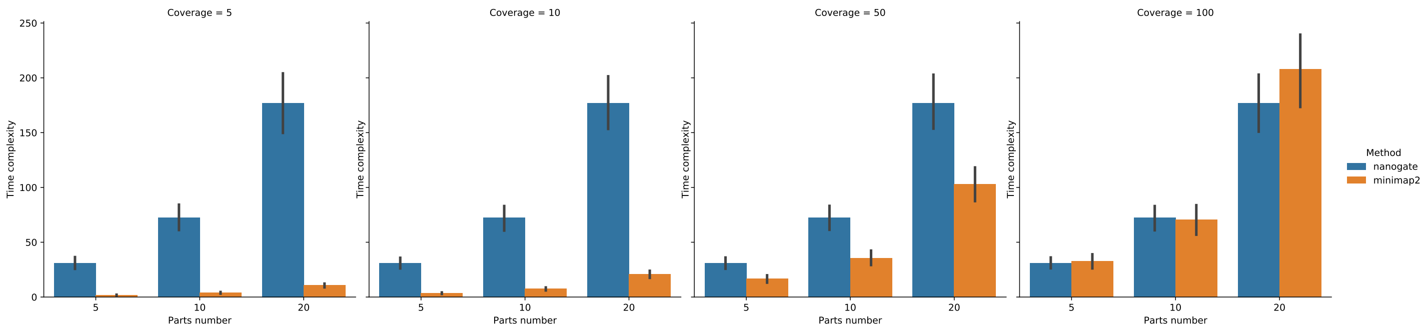

Figure 41: **Nanogate and Minimap2 time comparison plot.** This plot shows, for parts similarity$= 90\%$, the time complexity comparison between Nanogate and Minimap2 using as query only constructs reads.

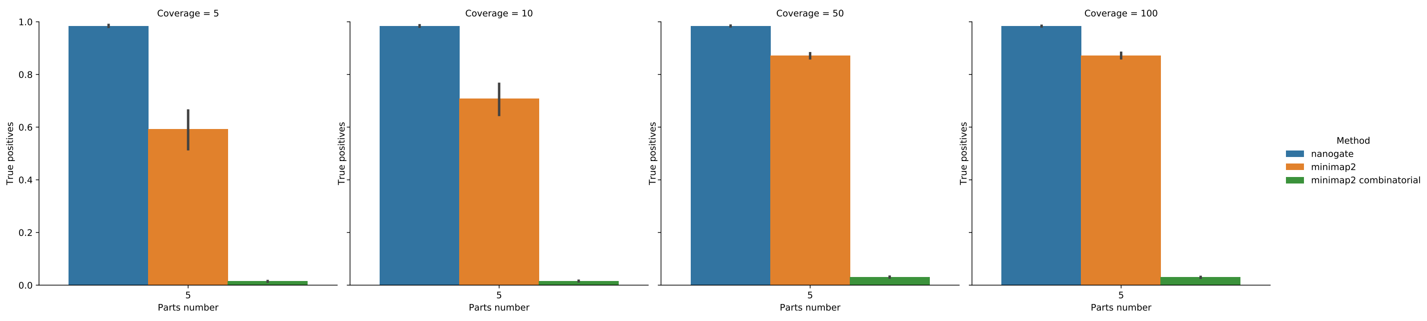

Figure 42: **Nanogate Minimap2 and combinatorial assembly true positives.** This plot shows, in absence of parts similarity, the true positives comparison between Nanogate, Minimap2 using as query only constructs reads, and Minimap2 applied to combinatorial assemblies. To simplify the analysis only 5 parts constructs are taken into account.

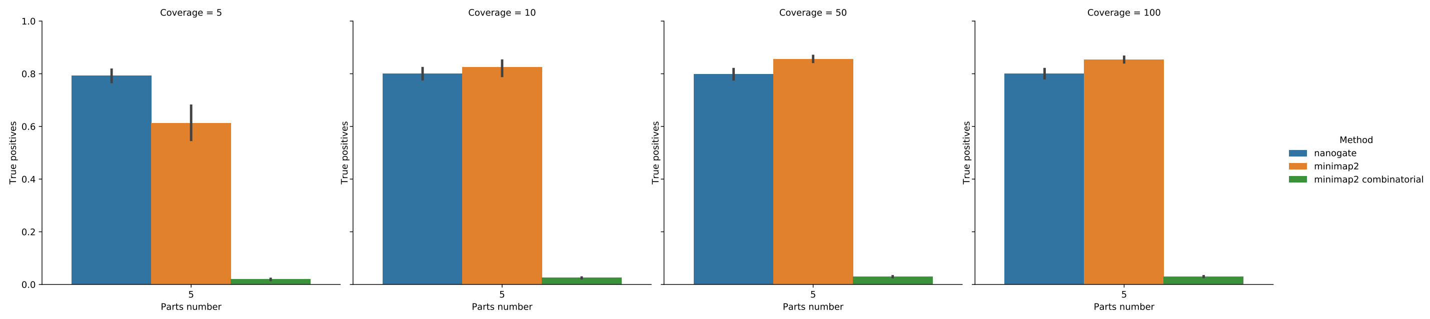

Figure 43: **Nanogate Minimap2 and combinatorial assembly true positives.** This plot shows, for parts similarity $= 90\%$, the true positives comparison between Nanogate, Minimap2 using as query only constructs reads, and Minimap2 applied to combinatorial assemblies. To simplify the analysis only 5 parts constructs are taken into account.

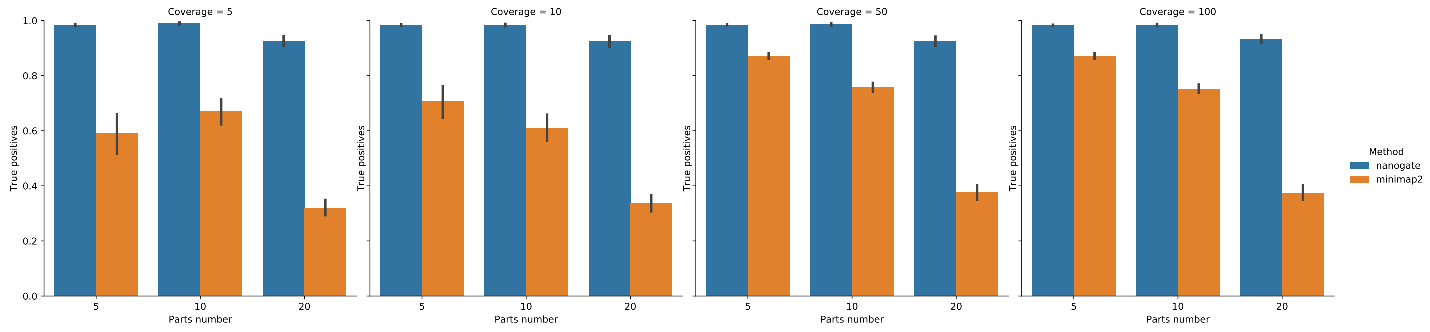

Figure 44: **Nanogate Minimap2 true positives.** This plot shows, in absence of parts similarity, the true positives comparison between Nanogate and Minimap2 using as query only constructs reads.

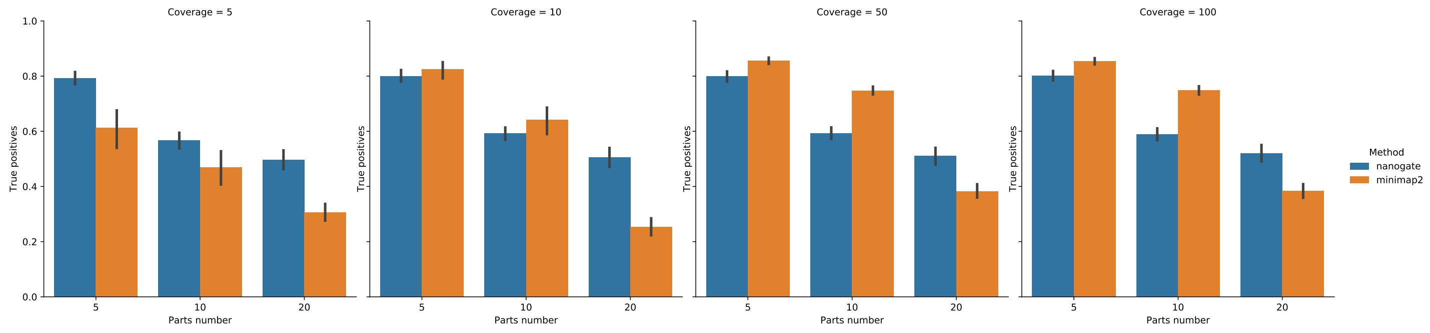

Figure 45: **Nanogate Minimap2 true positives.** This plot shows, for parts similarity$= 90\%$, the true positives comparison between Nanogate and Minimap2 using as query only constructs reads.

5.6 V4.5.0

Results in this section can be obtained by running the following pipeline:

```
project: ”bad_reads_sim_00”
genbank: ”yeast_chromosome_xiv.gb”
run: 10

generate_testbeds:
    cds_number : 50
    constructs_number: [10,50,100]
    parts : [5,10,20]
    constructs_similarity : 0.0
    parts_similarity : 0.0
    mutation_method : ”last”

badreads:
    quantity : [5x,10x,50x,100x]
    error_model: ”nanopore”
    qscore_model: ”nanopore”
    glitches: ”1000,100,100”
    junk_reads: 5
    random_reads: 5
    chimeras: 10
    identity: 75,90,8
    start_adapter_seq: '␣""␣'
    end_adapter_seq: '␣""␣'

combinatorial_assembly:
    max_parts: 5

nanogate:
    error_rate : [”001”,”010”,”030”]
    jaccard_thresholds: [”00”,”002”,”004”,”006”,”007”,”008”,”01”]
```

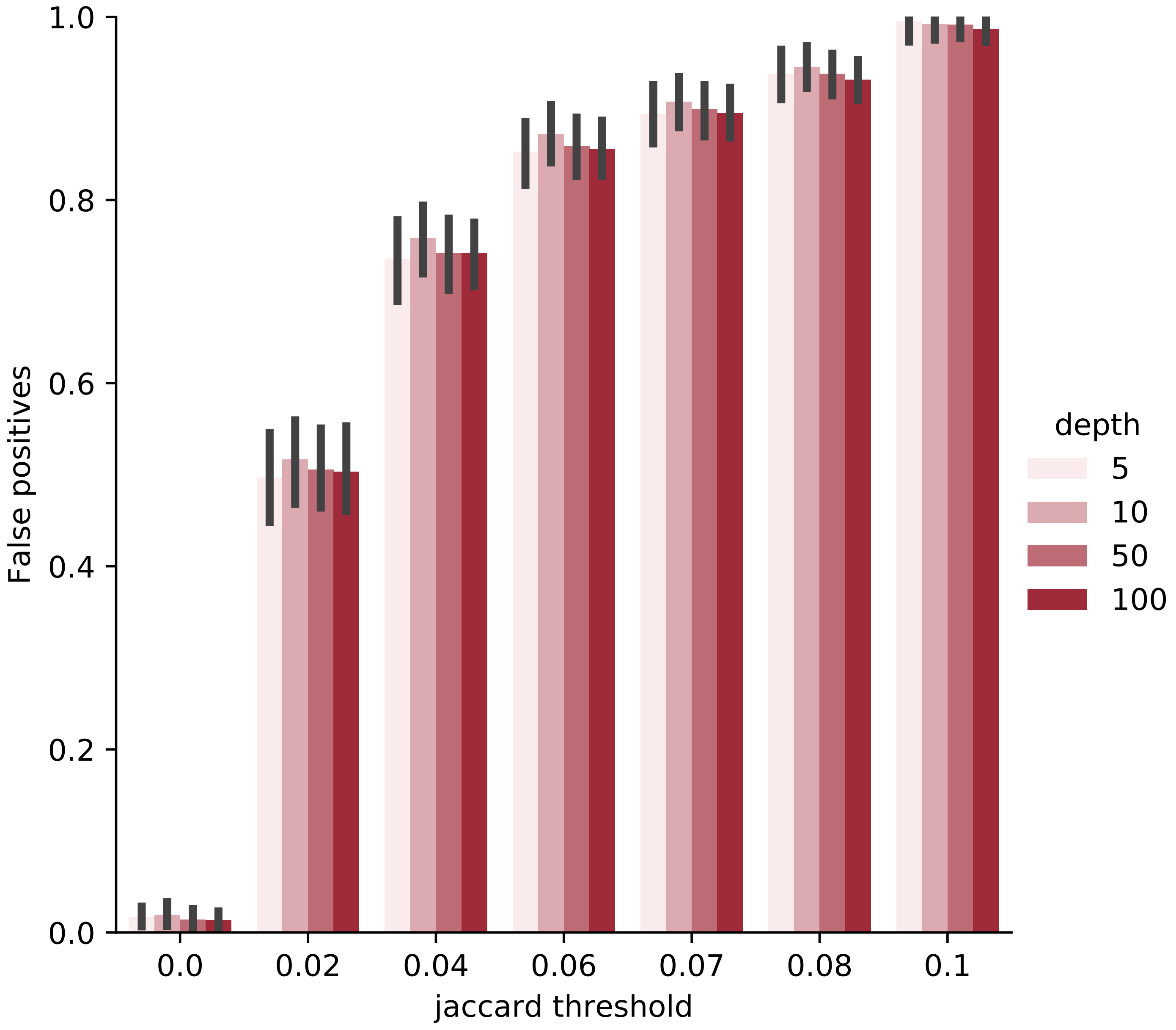

Figure 46: **Nanogate true positives.** This plot shows, in absence of parts similarity, the false positives rate. To evaluate false positives only junk reads and random reads are taken into consideration. A false positive is evaluated as the ratio p_0/br, where p_0 is number of junk or random reads with no part assigned and br is the number of the overall number of junk and random reads

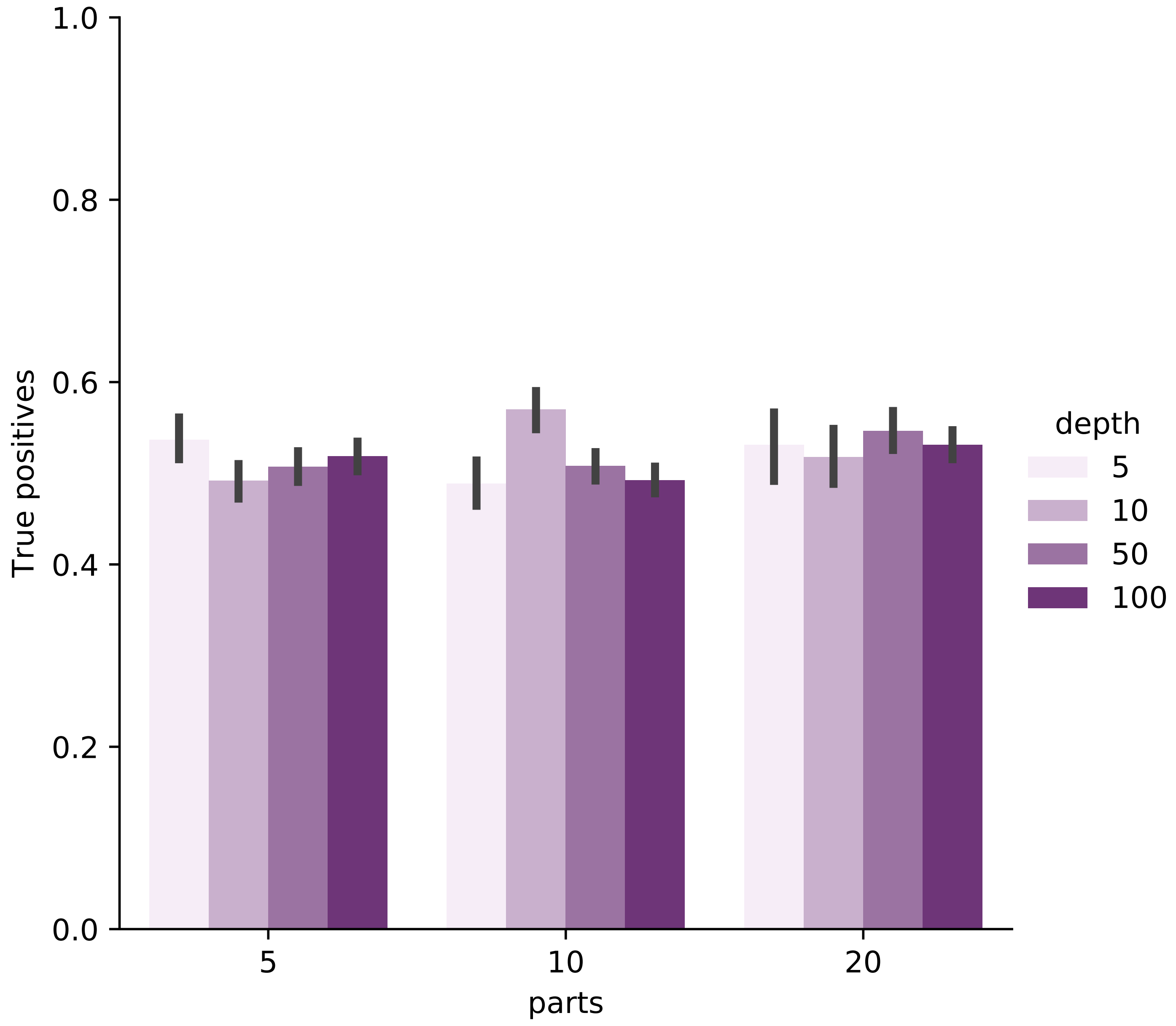

Figure 47: **Nanogate true positives.** This plot shows, in absence of parts similarity, the true positives rate when Nanogate is applied to reads with a bad quality

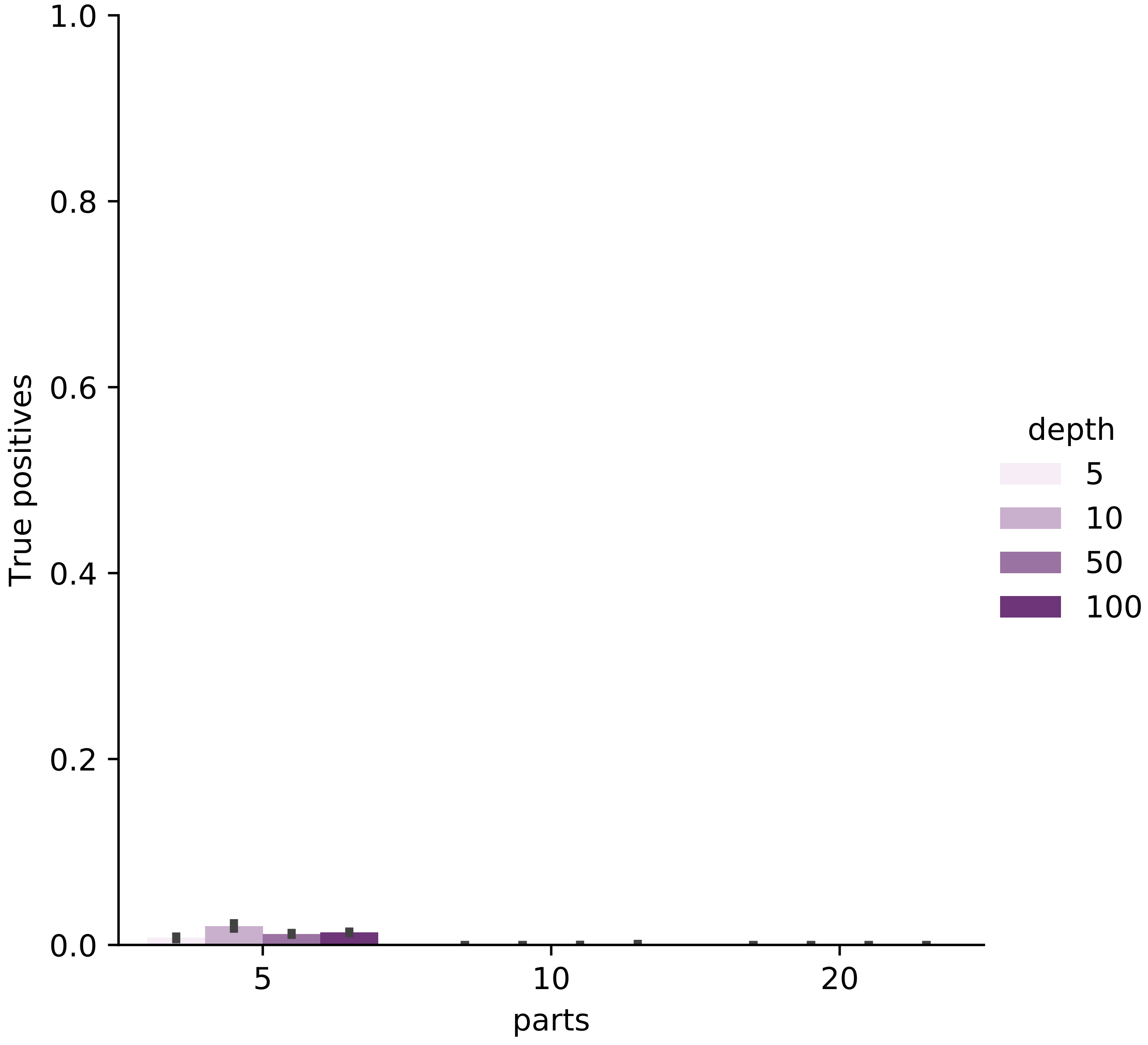

Figure 48: **Nanogate true positives.** This plot shows the true positives rate when Nanogate is applied to reads with a bad quality and the parts similarity $= 90\%$

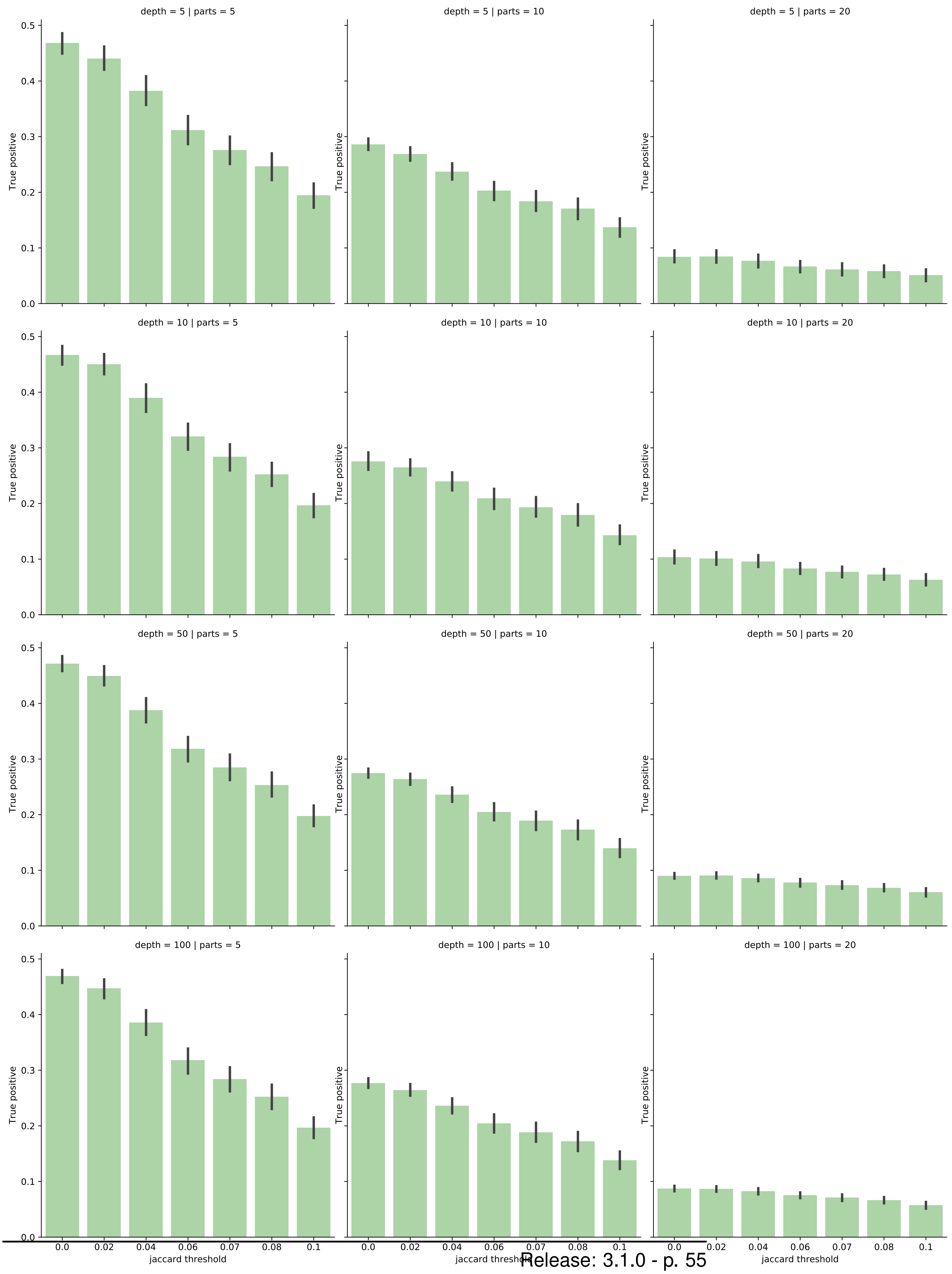

Figure 49: **Nanogate true positives.** This plot shows, in absence of parts similarity, the true positives rate when Nanogate is applied to reads with a bad quality.Here all reads who pass the Nanogate filter are considered not only those with a length $>= 99\%$ of the original construct length

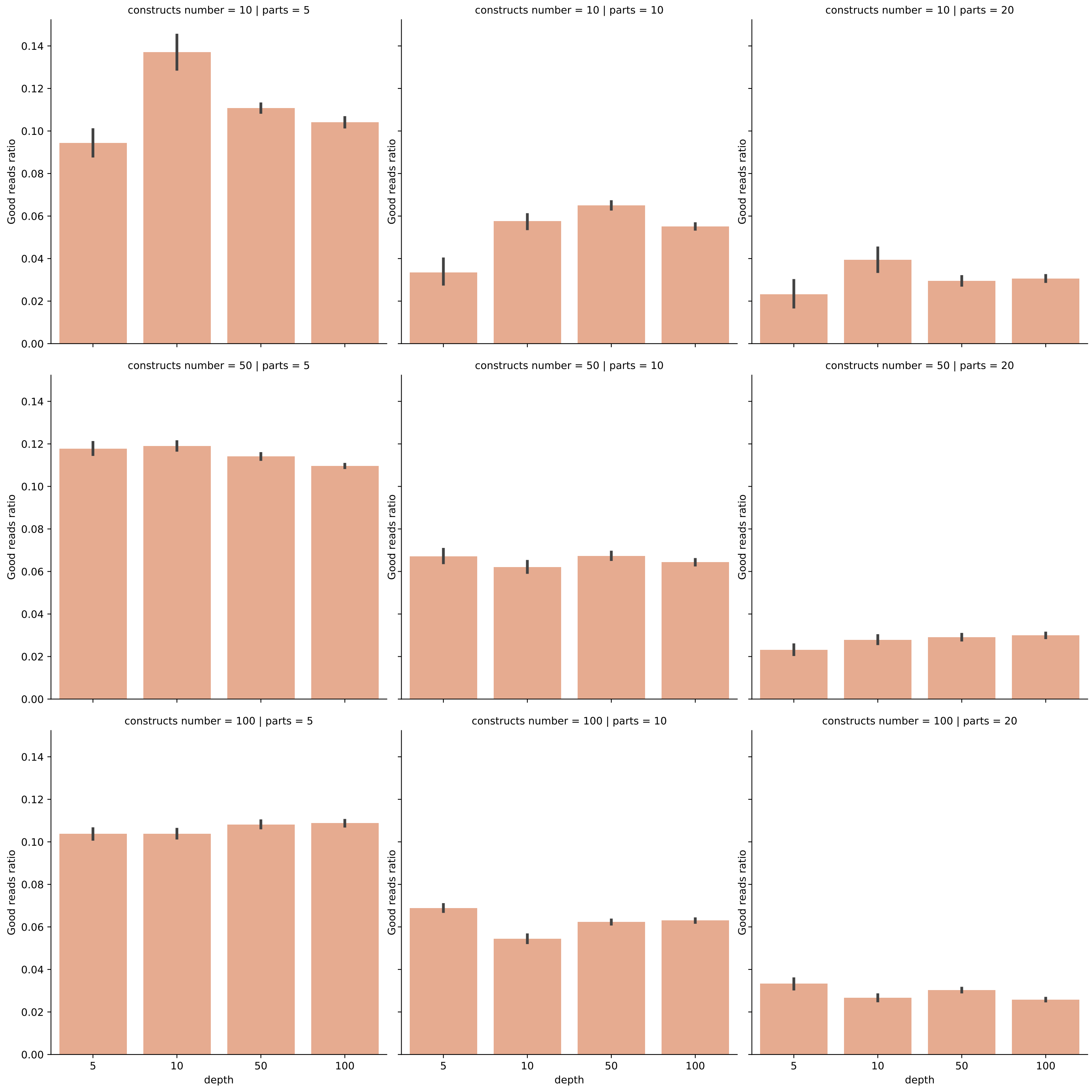

Figure 50: **Reads quality.** This plot shows the ratio $99r/r$, where $99r$ is the amount read with a length $>= 99\%$ of the original construct length, and r is the amount of all read analysed

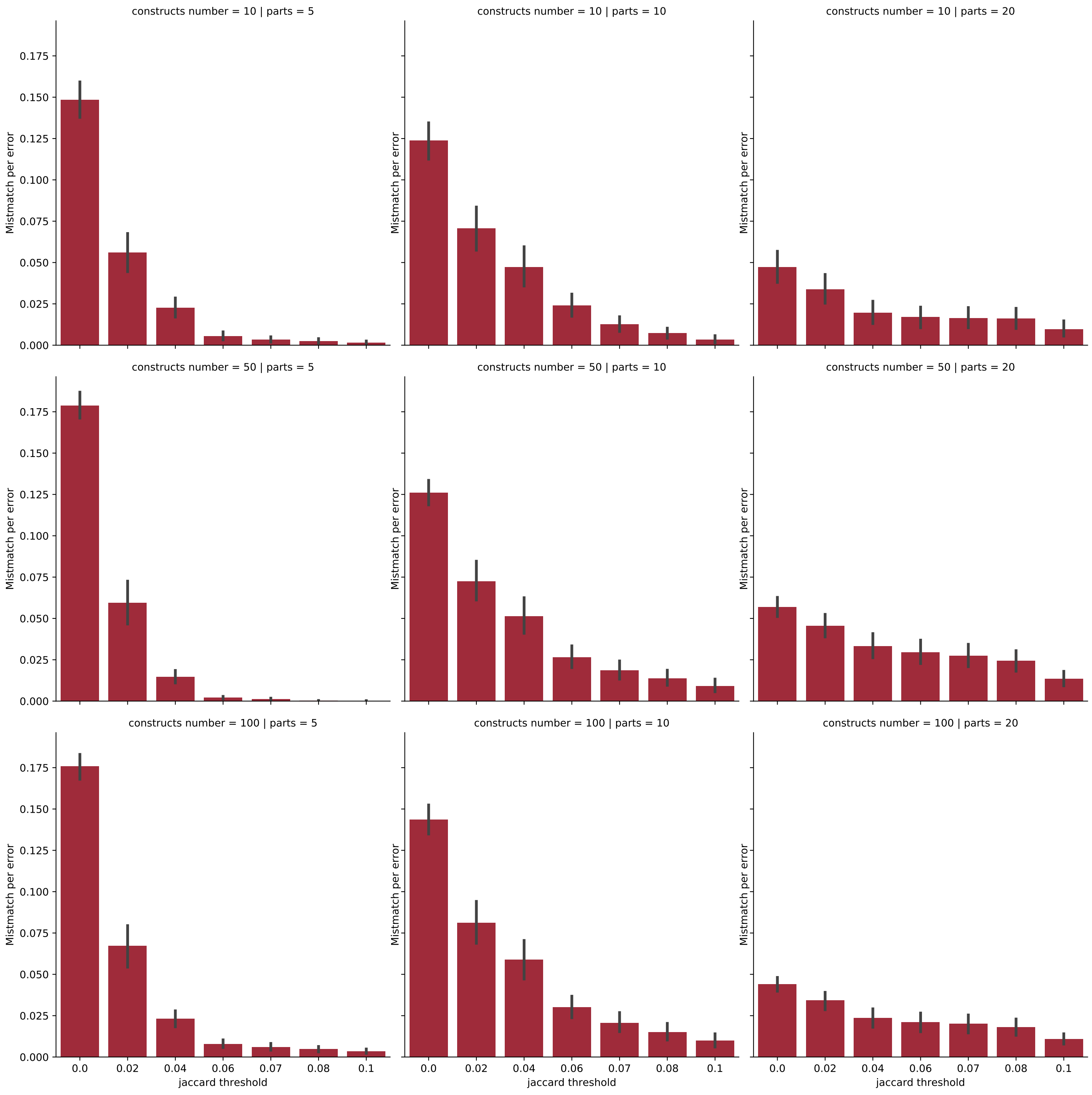

Figure 51: **Mismatch error.** This plot shows the ratio m/e, where m is amount of reads including at least of mismatch and e is the amount of reads analysed incorrectly. This plot shows how much of the Nanogate error is due to a part assigned incorrectly.

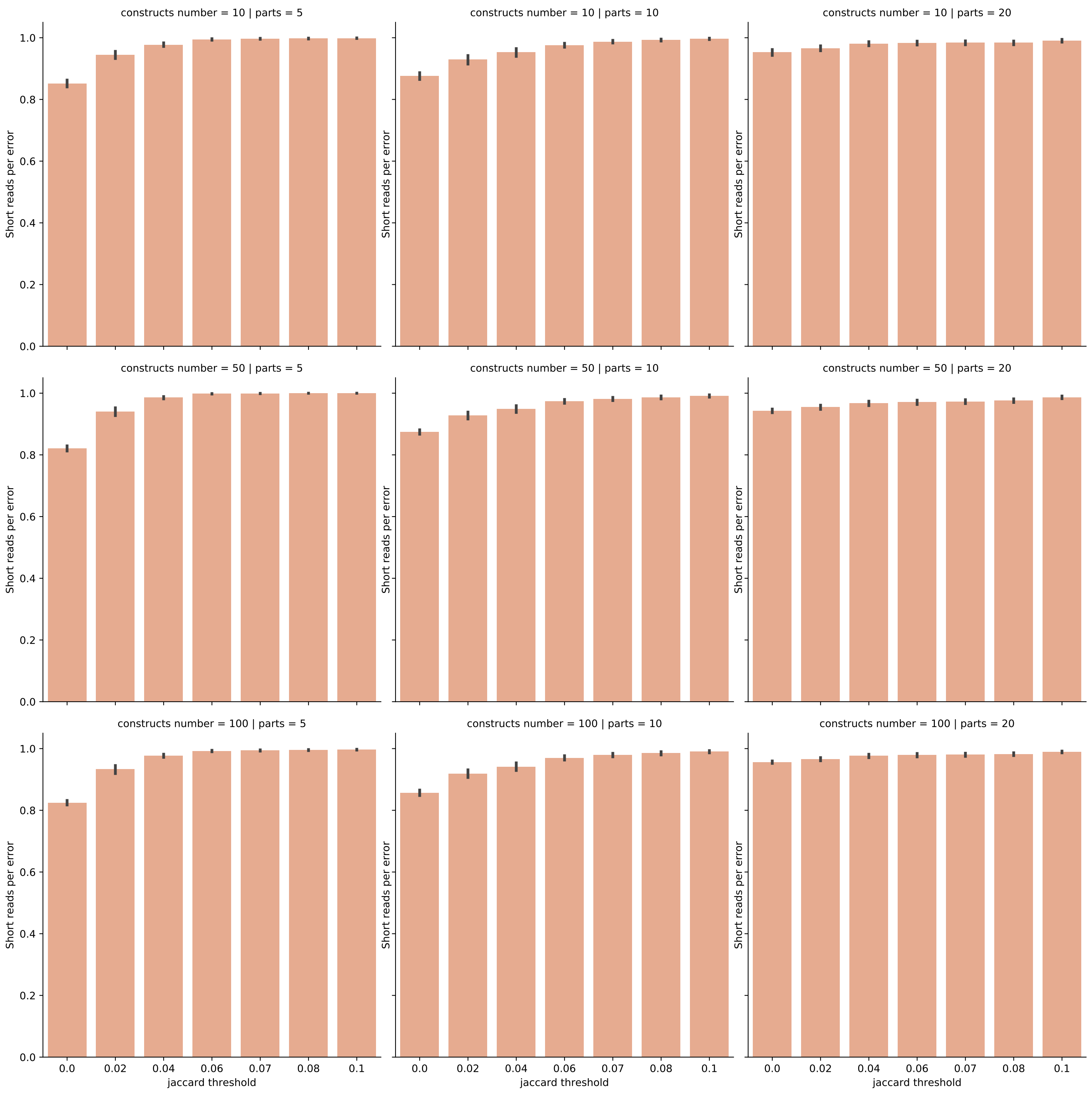

Figure 52: **Reads error.** This plot shows the ratio er/e, where er is the amount of reads including a different number of parts compared to the construct from which they derived.e is the amount of reads analysed incorrectly. These plot shows how much of the error is due to reads longer or shorter than the original construct, and as consequence they include more or less parts

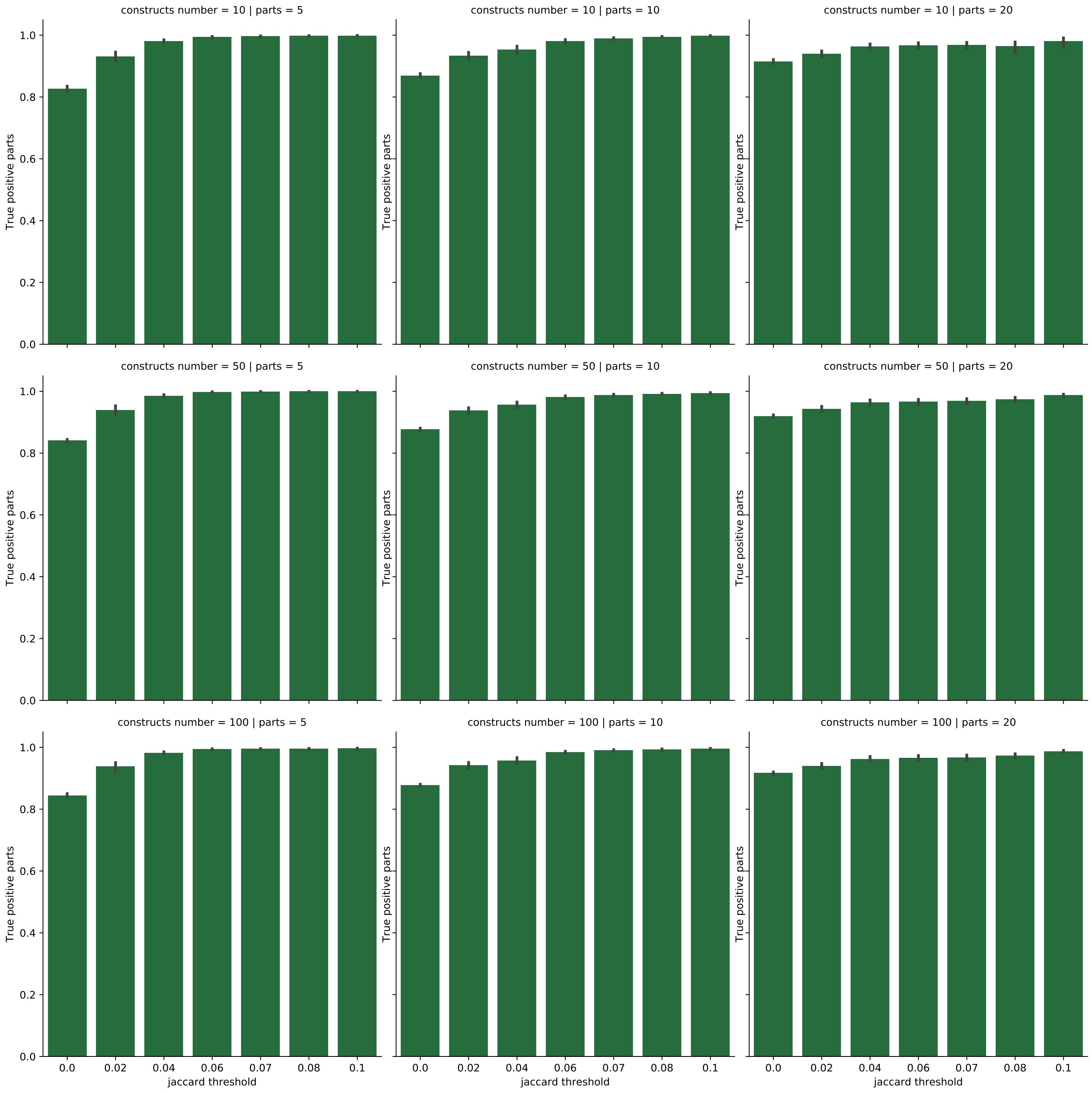

Figure 53: **Parts true positives.** This plot shows the ratio pc/p, where pc is the amount of parts correctly called and p is the amount of all parts analysed.This plot show how many parts have been assigned correctly.The similarity between parts $= 0\%$ in this plot

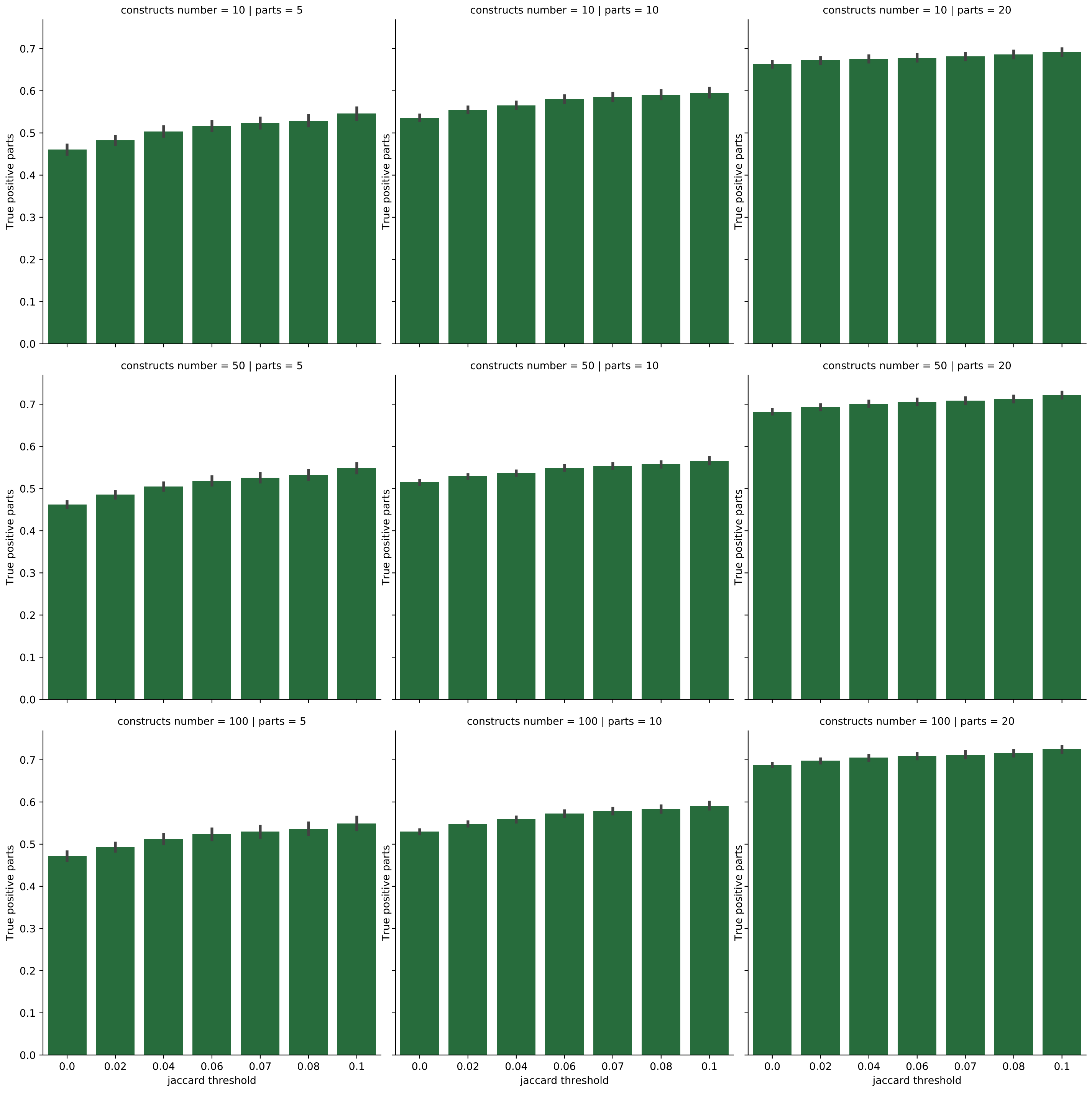

Figure 54: **Parts true positives.** This plot shows the ratio pc/p, where pc is the amount of parts correctly called and p is the amount of all parts analysed.This plot show how many parts have been assigned correctly.The similarity between parts $= 90\%$ in this plot

5.7 V4.6.0

```
project: "bad_reads_sim_00"
genbank: "yeast_chromosome_xiv.gb"
run: 10

generate_testbeds:
    cds_number : 50
    constructs_number: [10,50,100]
    parts : [5,10,20]
    constructs_similarity : 0.0
    parts_similarity : 0.0
    mutation_method : "last"

badreads:
    quantity : [5x,10x,50x,100x]
    error_model: "nanopore"
    qscore_model: "nanopore"
    glitches: "1000,100,100"
    junk_reads: 5
    random_reads: 5
    chimeras: 10
    identity: 75,90,8
    start_adapter_seq: '␣""␣'
    end_adapter_seq: '␣""␣'

combinatorial_assembly:
    max_parts: 5

nanogate:
    error_rate : ["001","010","030"]
    jaccard_thresholds: ["00","002","004","006","007","008","01"]
```

5.7.1 Contaminant discovery

Random and junk reads are analysed separately from others, here I show the ratio

$$cd = \text{contaminants detected}/\text{all contaminants} \tag{1}$$

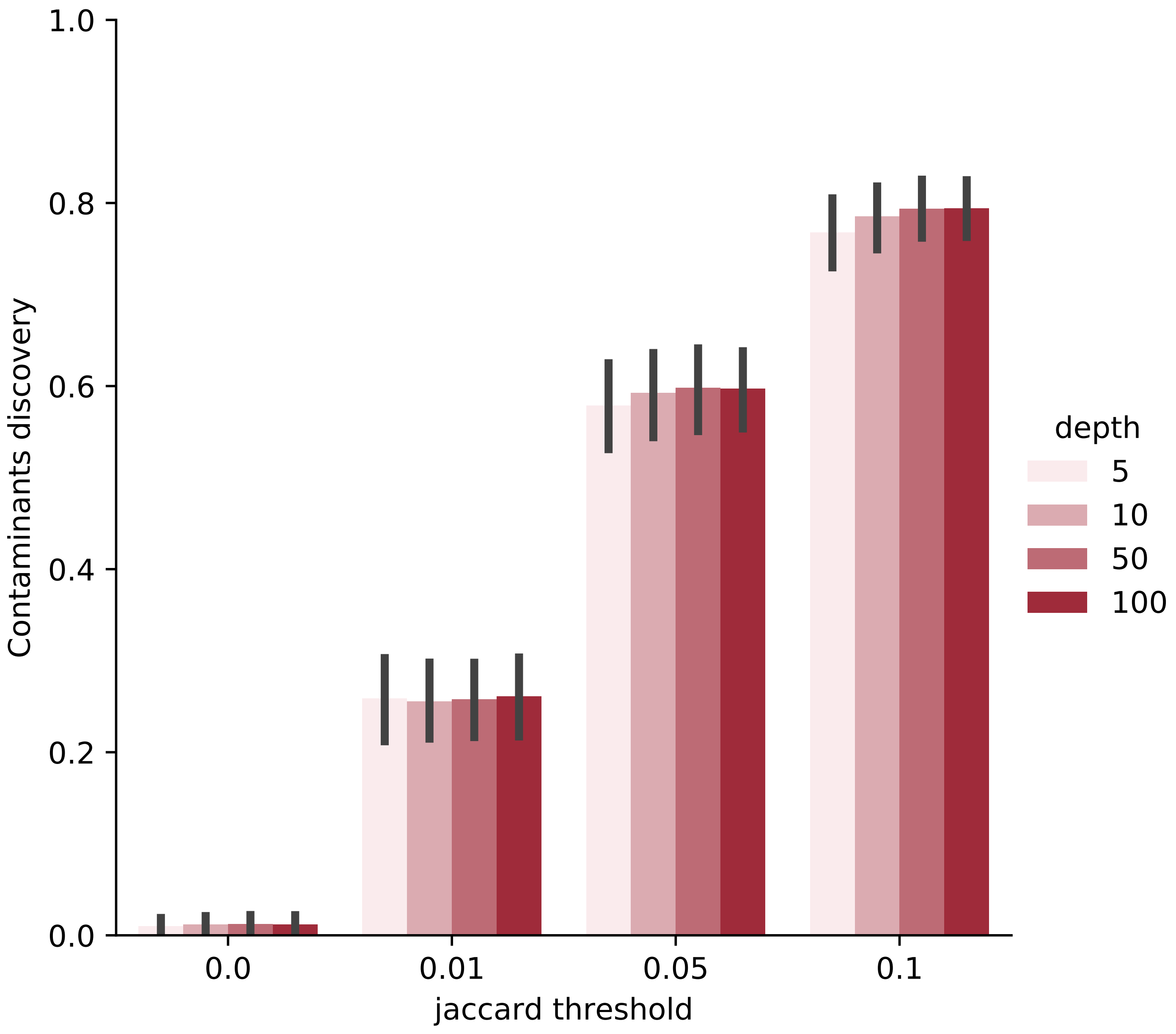

Figure 55: **Contaminants discovery.** This plot shows the ratio between the detected contaminants and all contaminants.We see how increasing the J_t, the contaminant discovery increase as well. Contaminants discovery is applied only on low quality reads, here in particular

5.7.2 Reads quality analysis

In this version of Nanogate we are analysing only pass reads. We mark reads as pass, if Nanogate assign them the preset number of parts. Here I show the difference in terms of reads analysed between the high quality setting and the low quality.

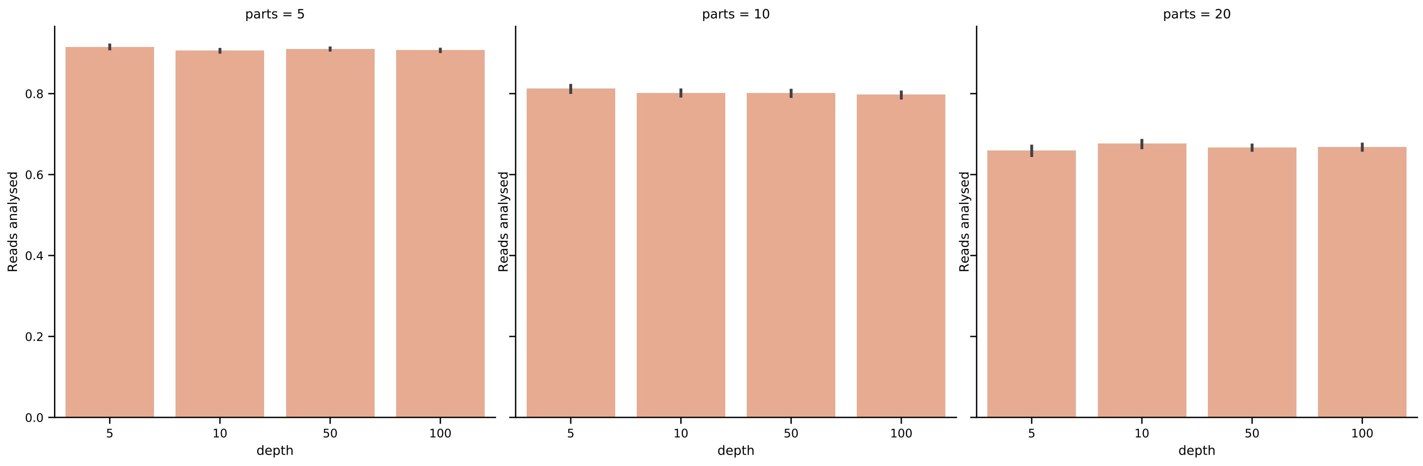

Figure 56: **High quality reads analysed .** This plot show ration of reads analysed who passed the Nanogate filter, here I measure the ratio between reads who passed the filter and are marked as pass and every reads who pass the filter. The analysis is performed on High quality reads and in absence of parts similarity.

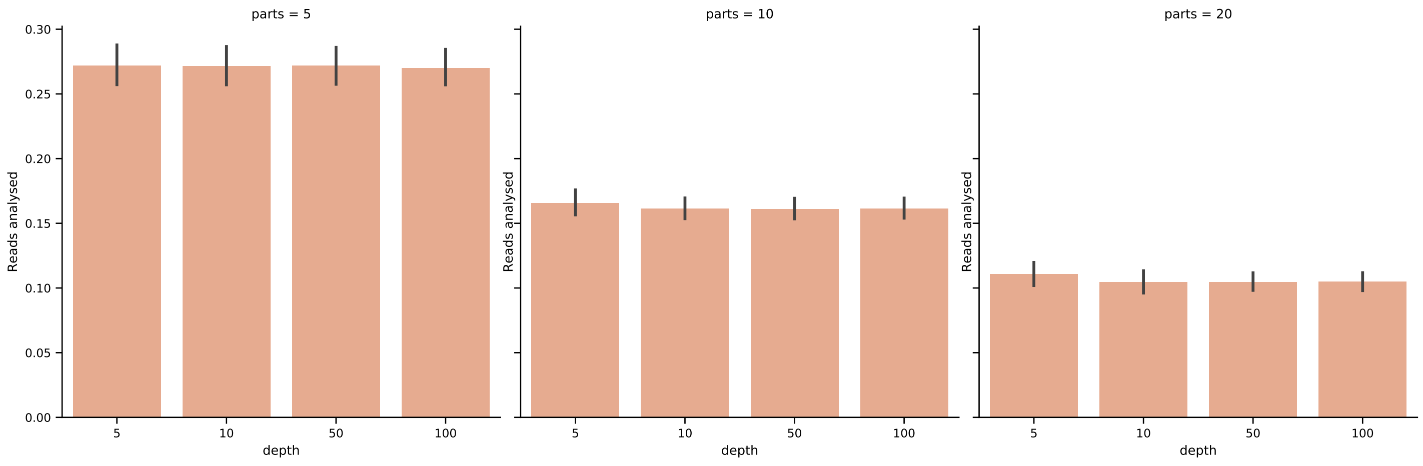

Figure 57: **Low quality reads analysed .** This plot show ration of reads analysed who passed the Nanogate filter, here I measure the ratio between reads who passed the filter and are marked as pass and every reads who pass the filter. The analysis is performed on Low quality reads and with parts similarity $= 90\%$.

5.7.3 Confusion matrix analysis

Here I analyse Nanogate precision and recall.
Positivity definition:

- **True Positive(TP)** read correctly analysed by Nanogate, where all parts are predicted correctly
- **False Positive(FP)** read analysed incorrectly, which composition defined by Nanogate matches the composition of another construct in the pool
- **False Negative(FN)** read analysed incorrectly, which composition defined by Nanogate matches none of the construct in the pool

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

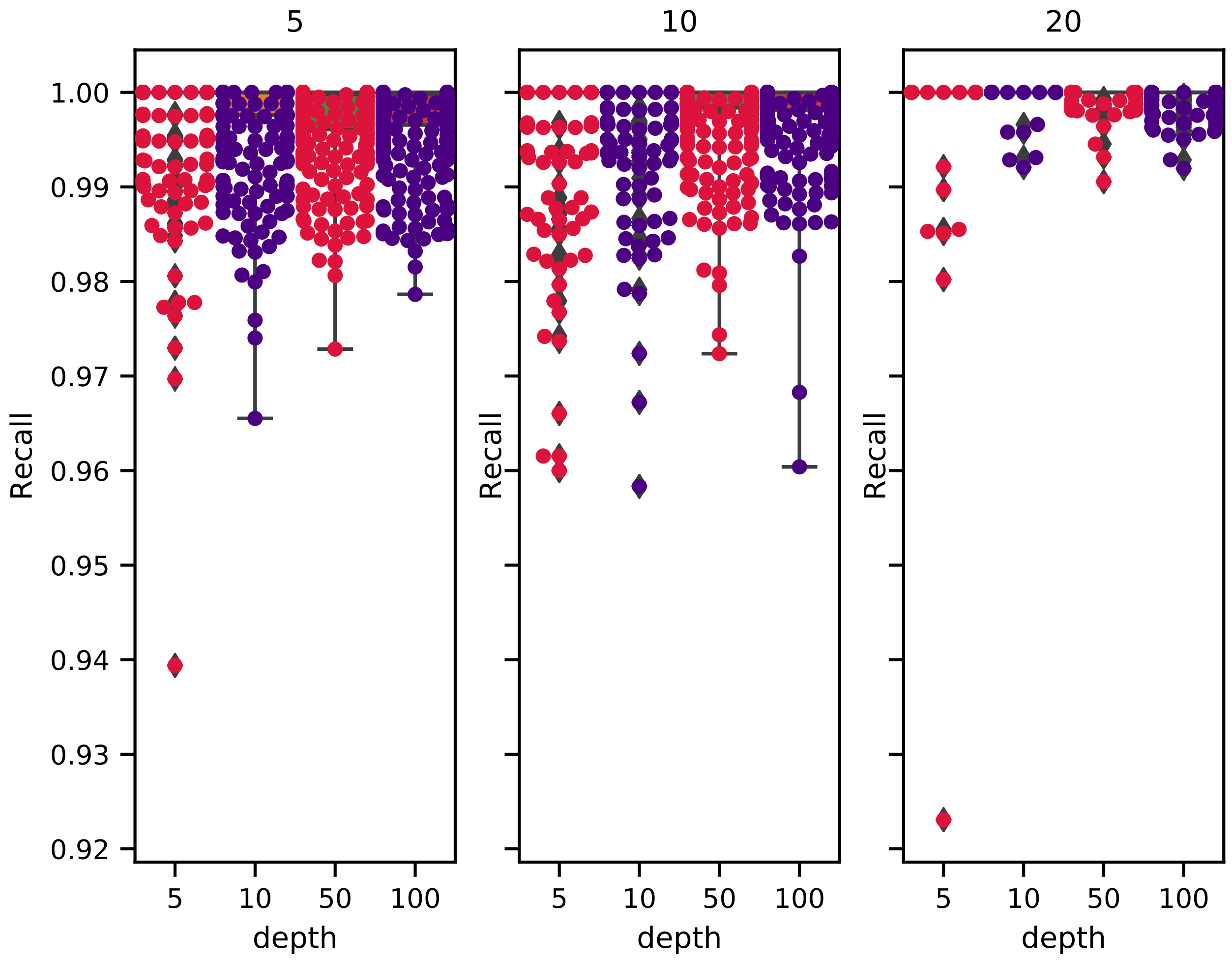

Figure 58: **High quality recall per result .** This plot shows the recall related to the high quality setting, in absence of parts similarity. The recall is evaluated on each each Nanogate output, each point represent the recall evaluated on a single output

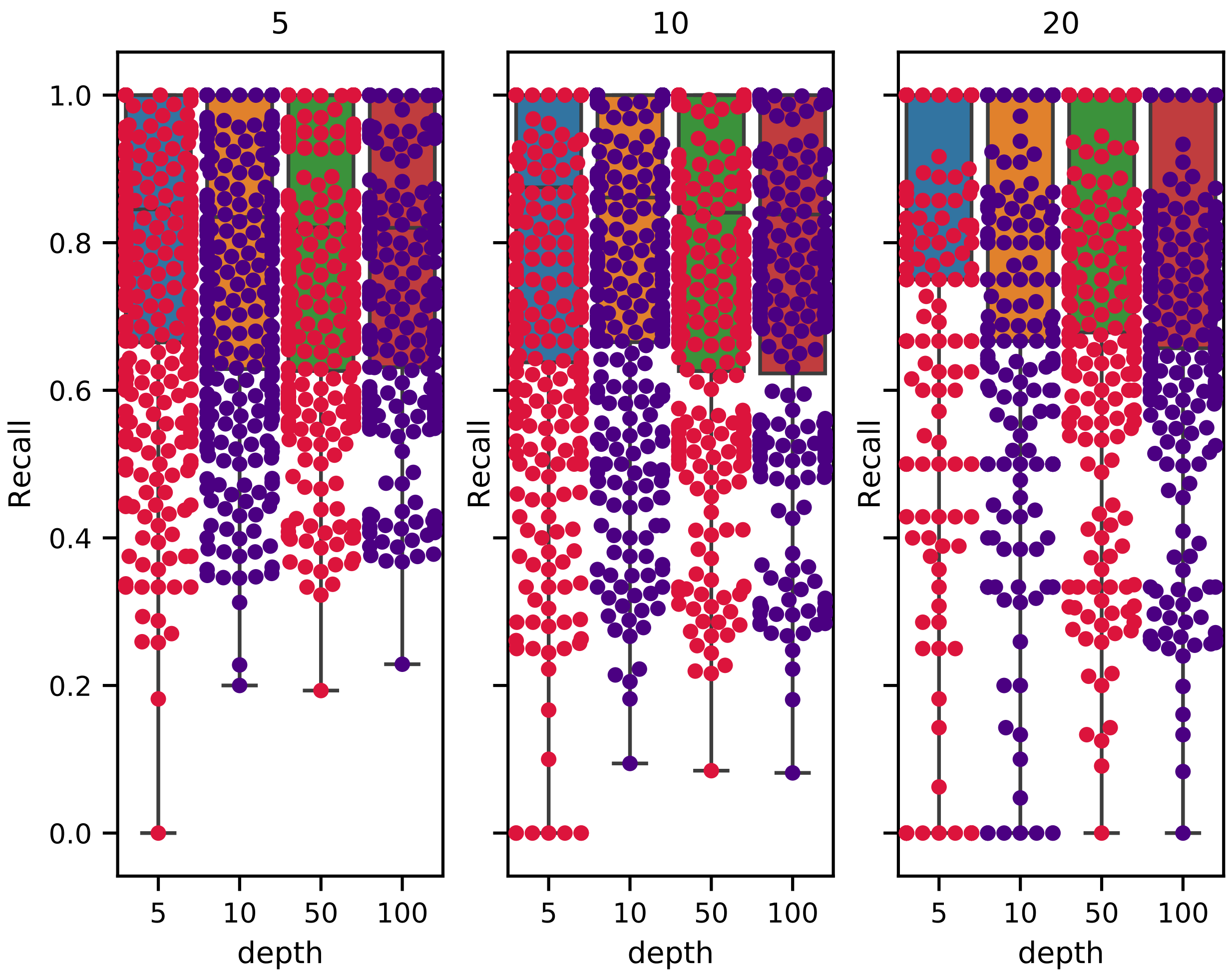

Figure 59: **Low quality recall per result similarity** 0% **.** This plot shows the recall related to the low quality setting, in absence of parts similarity. The recall is evaluated on each each Nanogate output, each point represent the recall evaluated on a single output

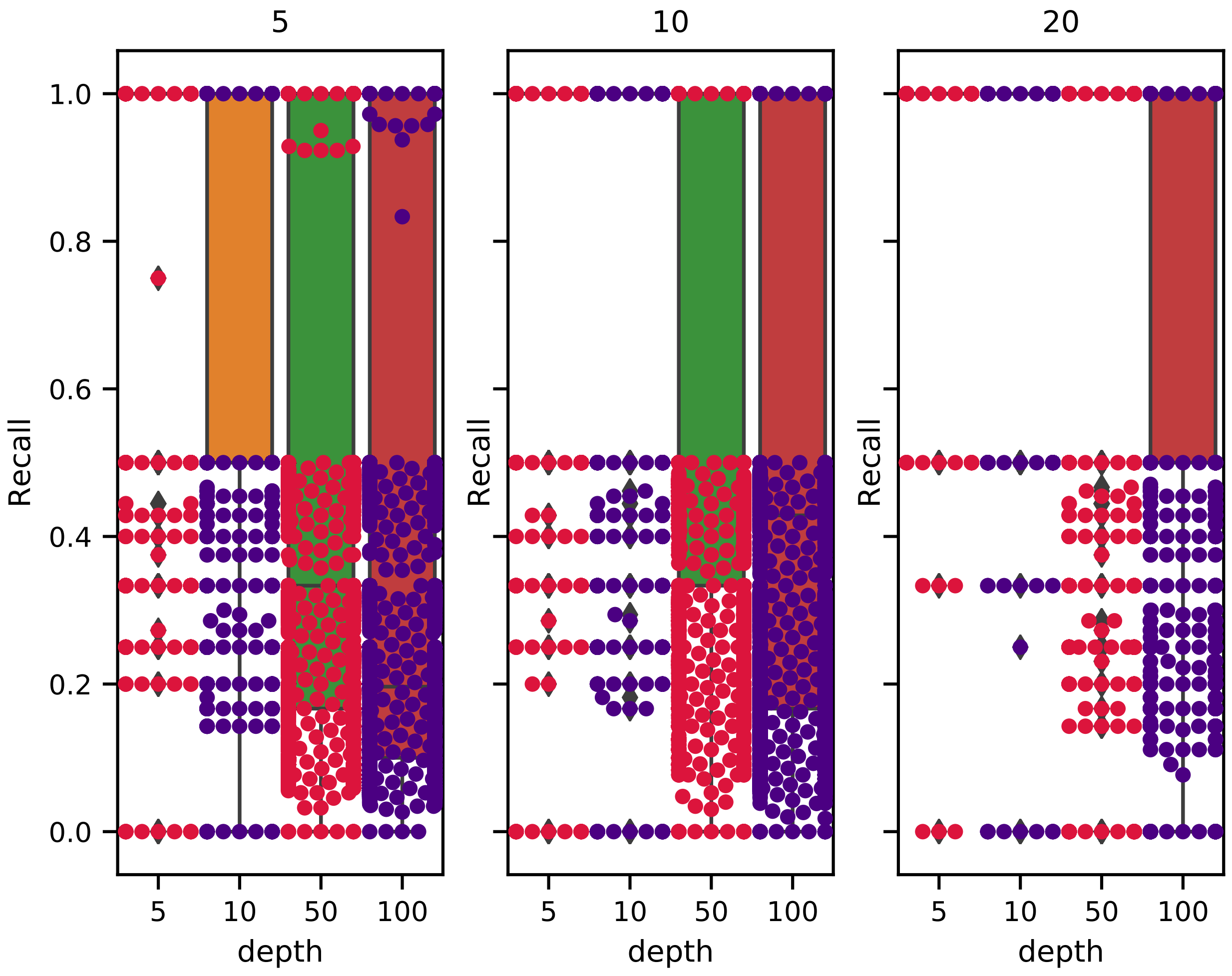

Figure 60: **Low quality recall per construct similarity** 0% **.** This plot shows the recall related to the low quality setting, in absence of parts similarity. The recall is evaluated on each construct, each point represent a construct

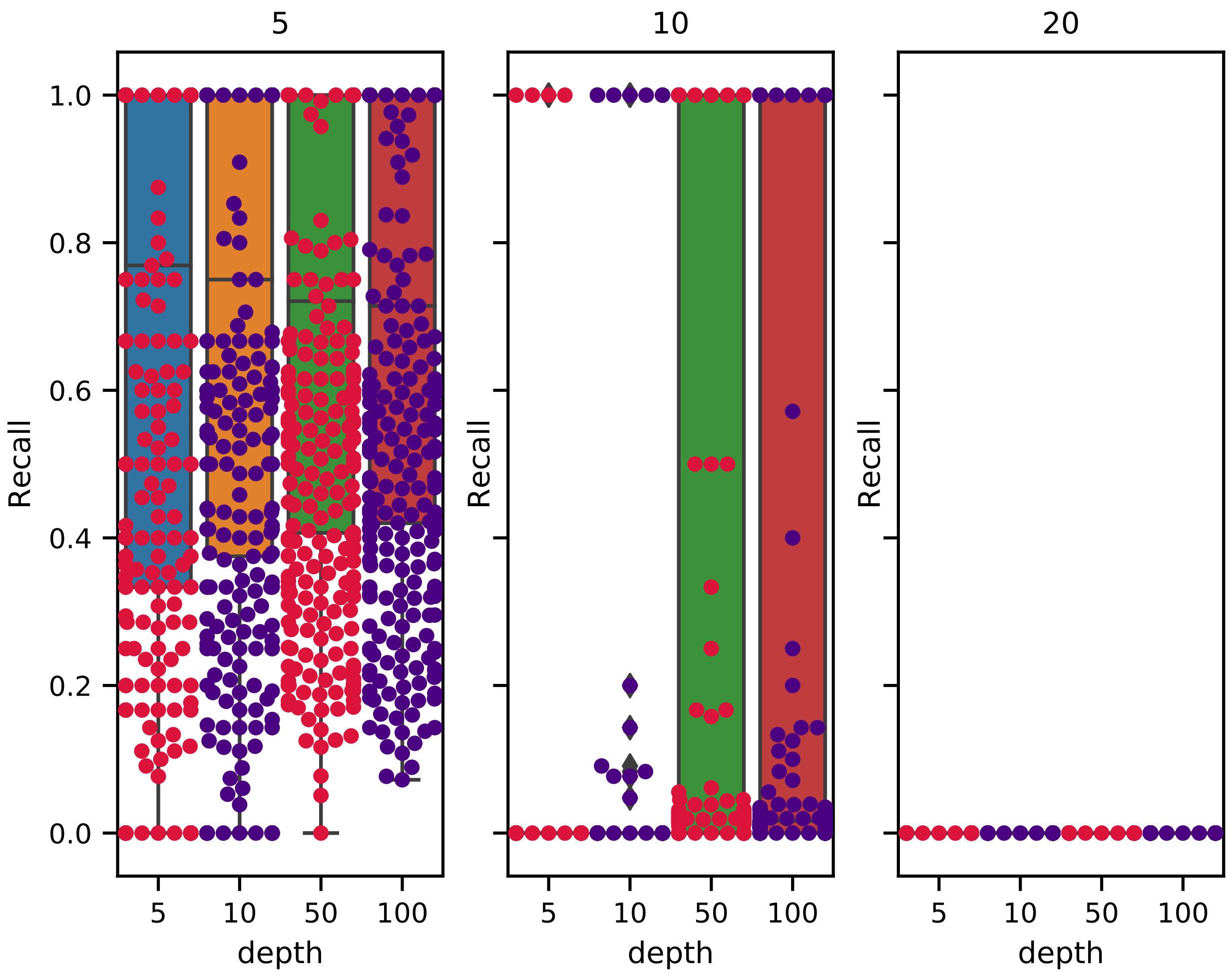

Figure 61: **Low quality recall per result similarity** 90% **.** This plot shows the recall related to the low quality setting, with parts similarity $= 90\%$. The recall is evaluated on each each Nanogate output, each point represent the recall evaluated on a single output

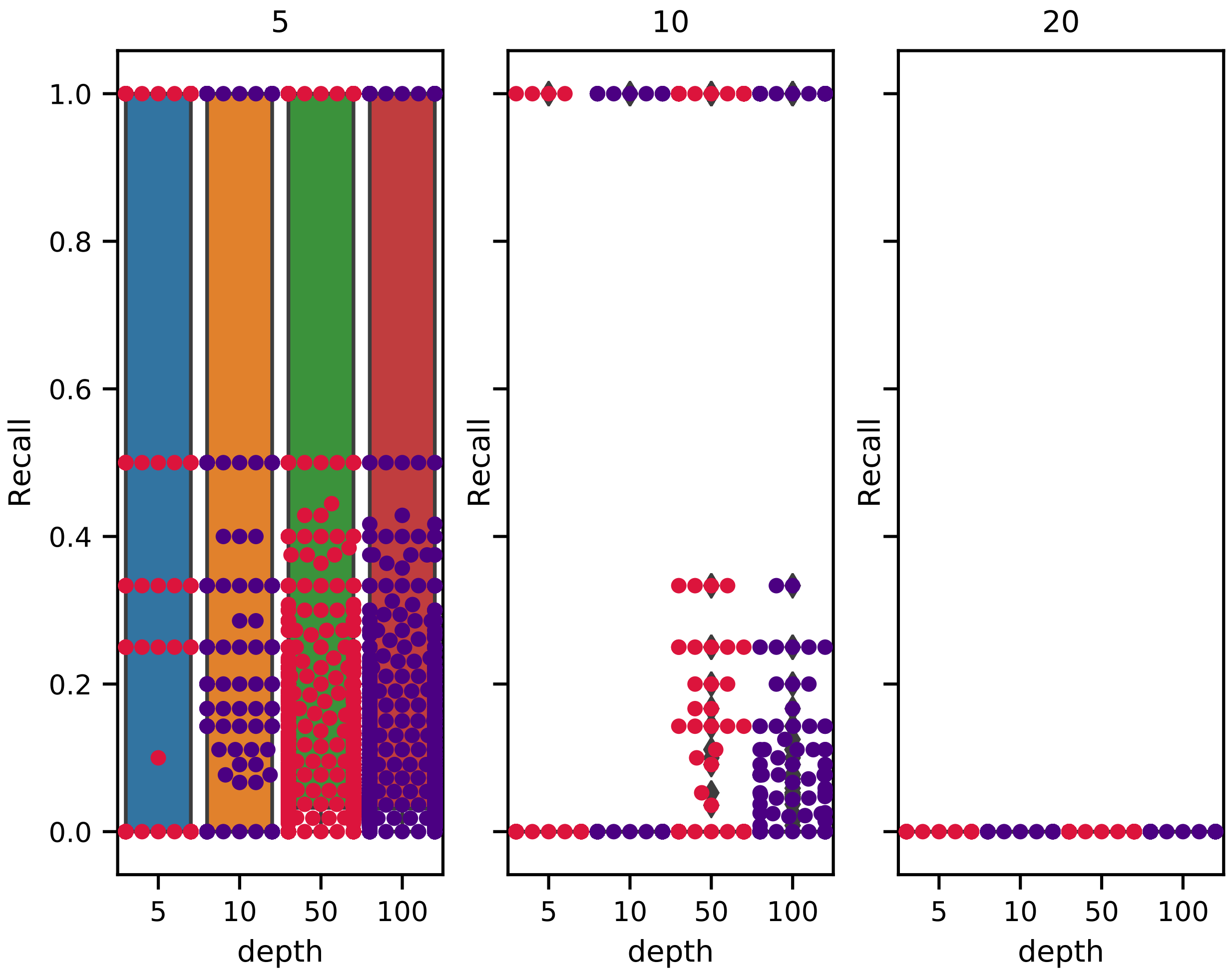

Figure 62: **Low quality recall per construct similarity** 90% **.** This plot shows the recall related to the low quality setting, with parts similarity $= 90\%$. The recall is evaluated on each construct, each point represent a construct

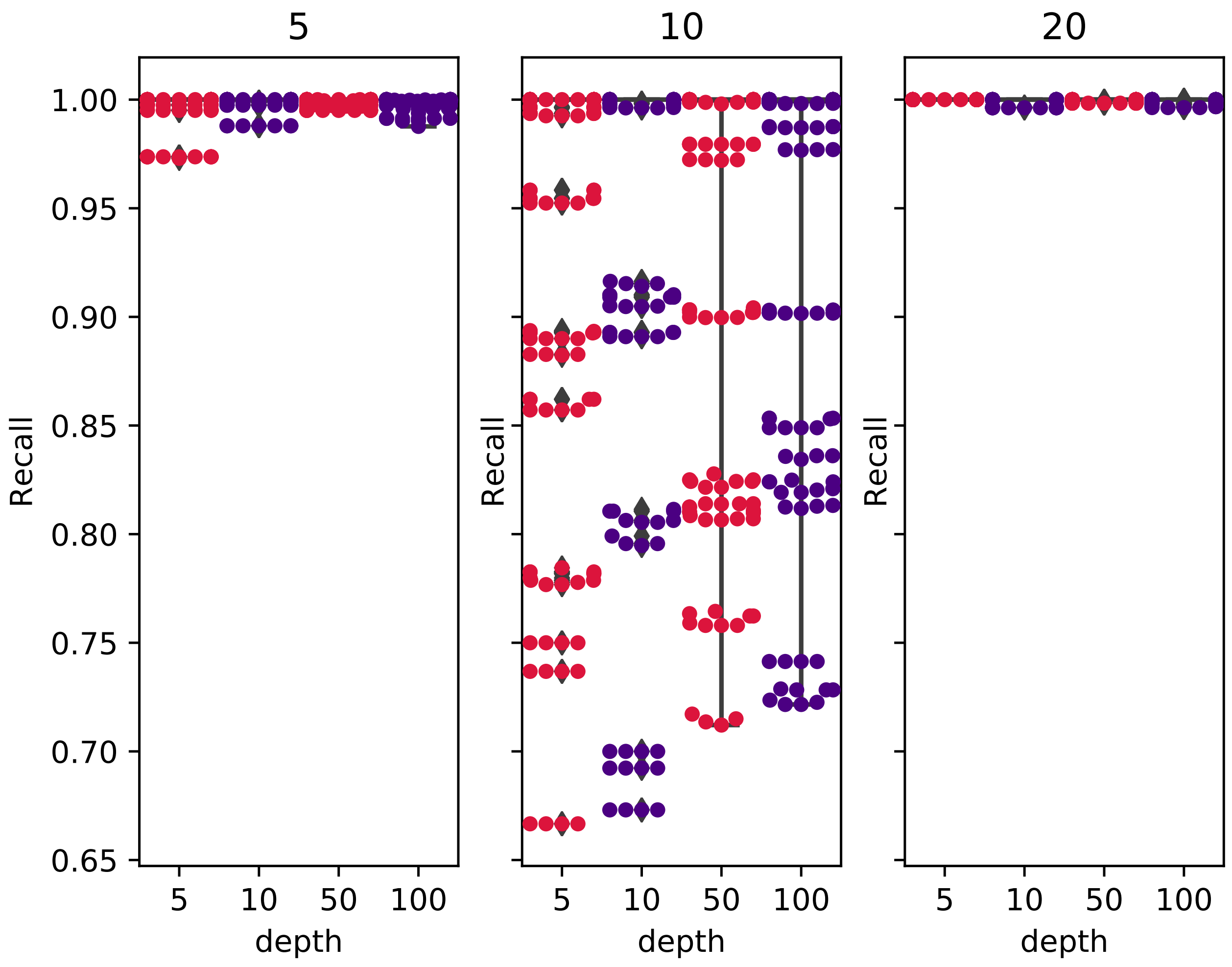

Figure 63: **High quality precision per result similarity** 0% **.** This plot shows the precision related to the high quality setting, with parts similarity $= 0\%$. The precision is evaluated on each each Nanogate output, each point represent the recall evaluated on a single output. (The plot is correct, there is a bug with labels, this plot shows the precision)

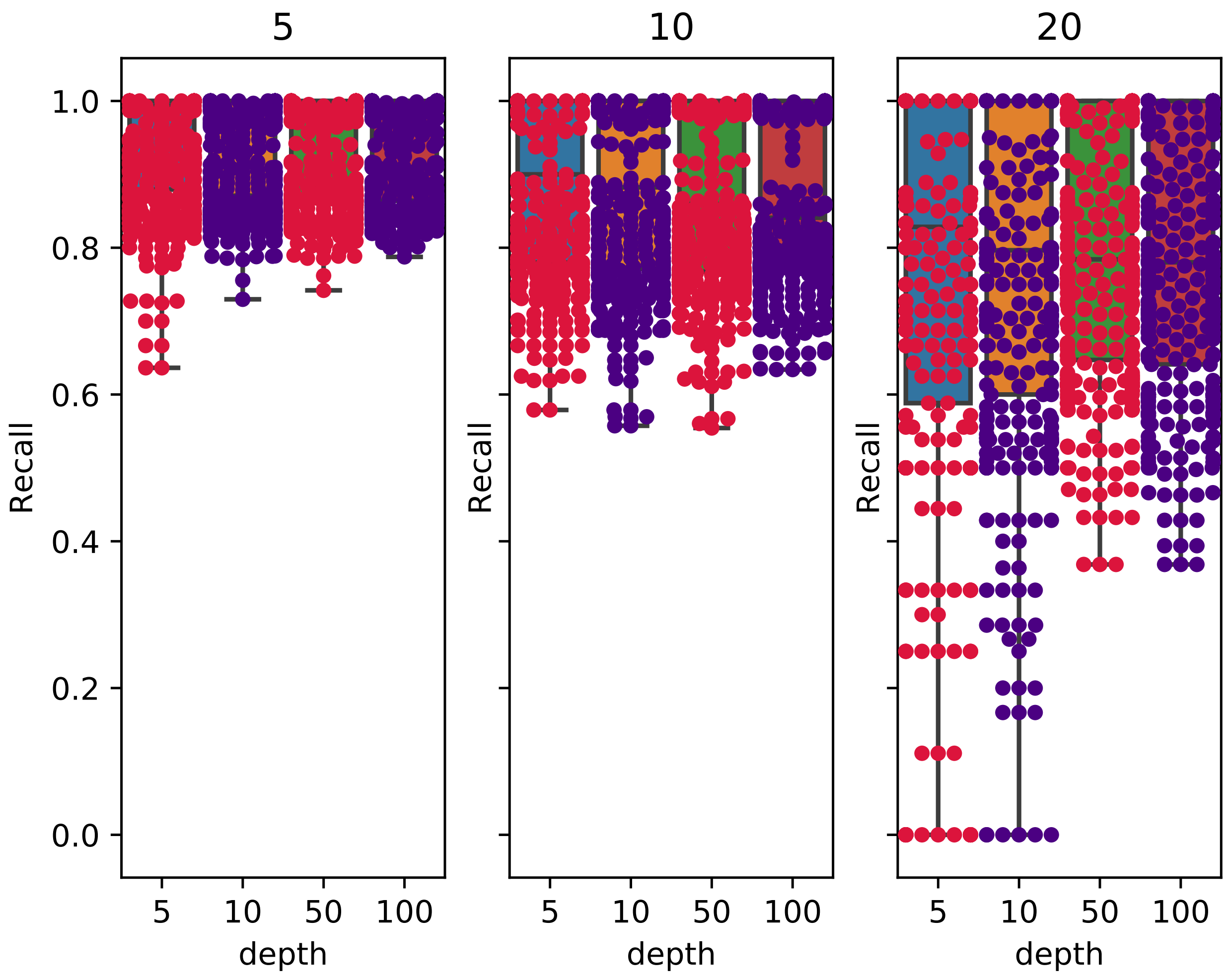

Figure 64: **Low quality precision per result similarity** 0% **.** This plot shows the precision related to the low quality setting, with parts similarity $= 0\%$. The precision is evaluated on each each Nanogate output, each point represent the recall evaluated on a single output. (The plot is correct, there is a bug with labels, this plot shows the precision)

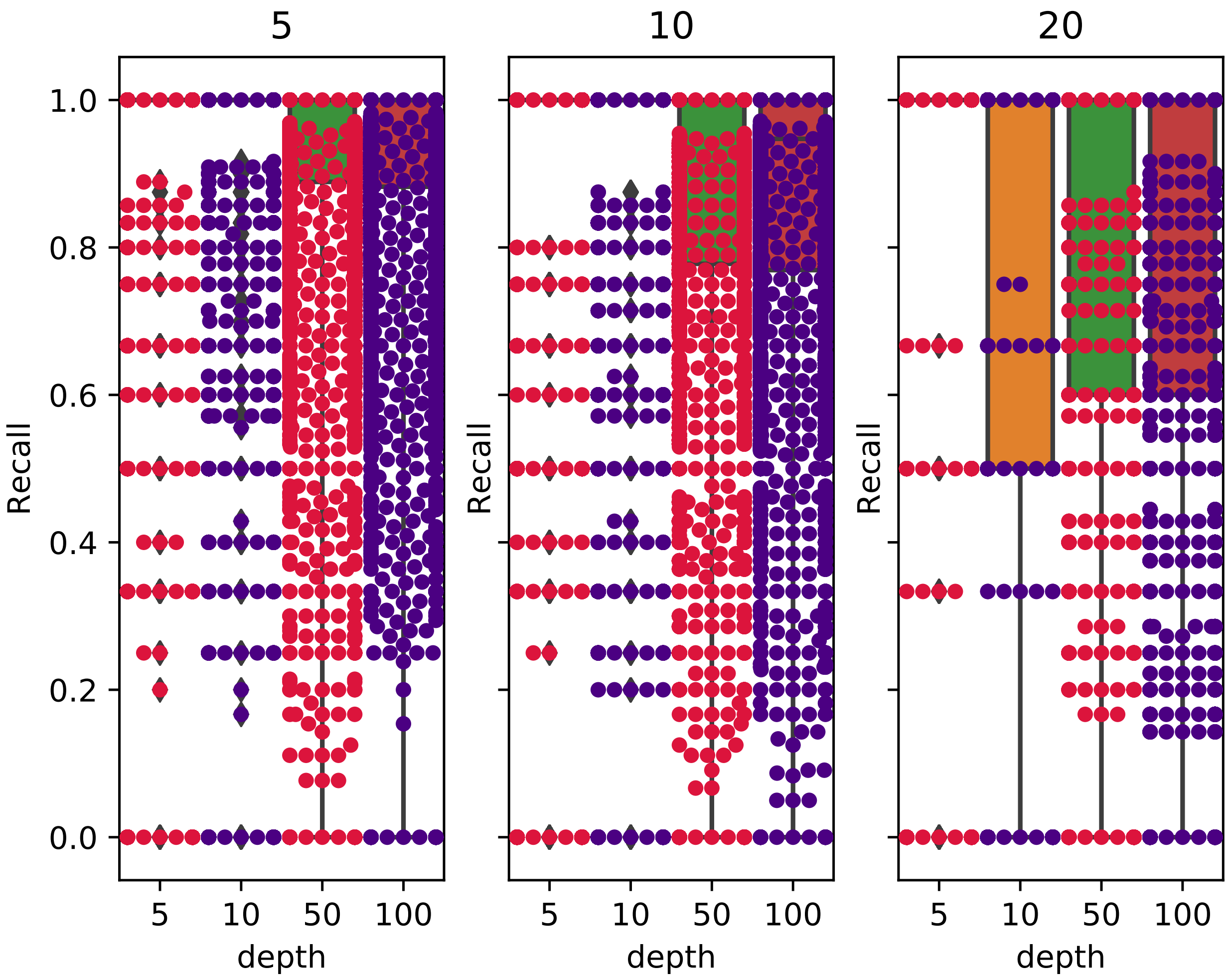

Figure 65: **Low quality precision per construct similarity** 0% **.** This plot shows the precision related to the low quality setting, with parts similarity $= 0\%$. The precision is evaluated on each construct, each point represent a construct. (The plot is correct, there is a bug with labels, this plot shows the precision)

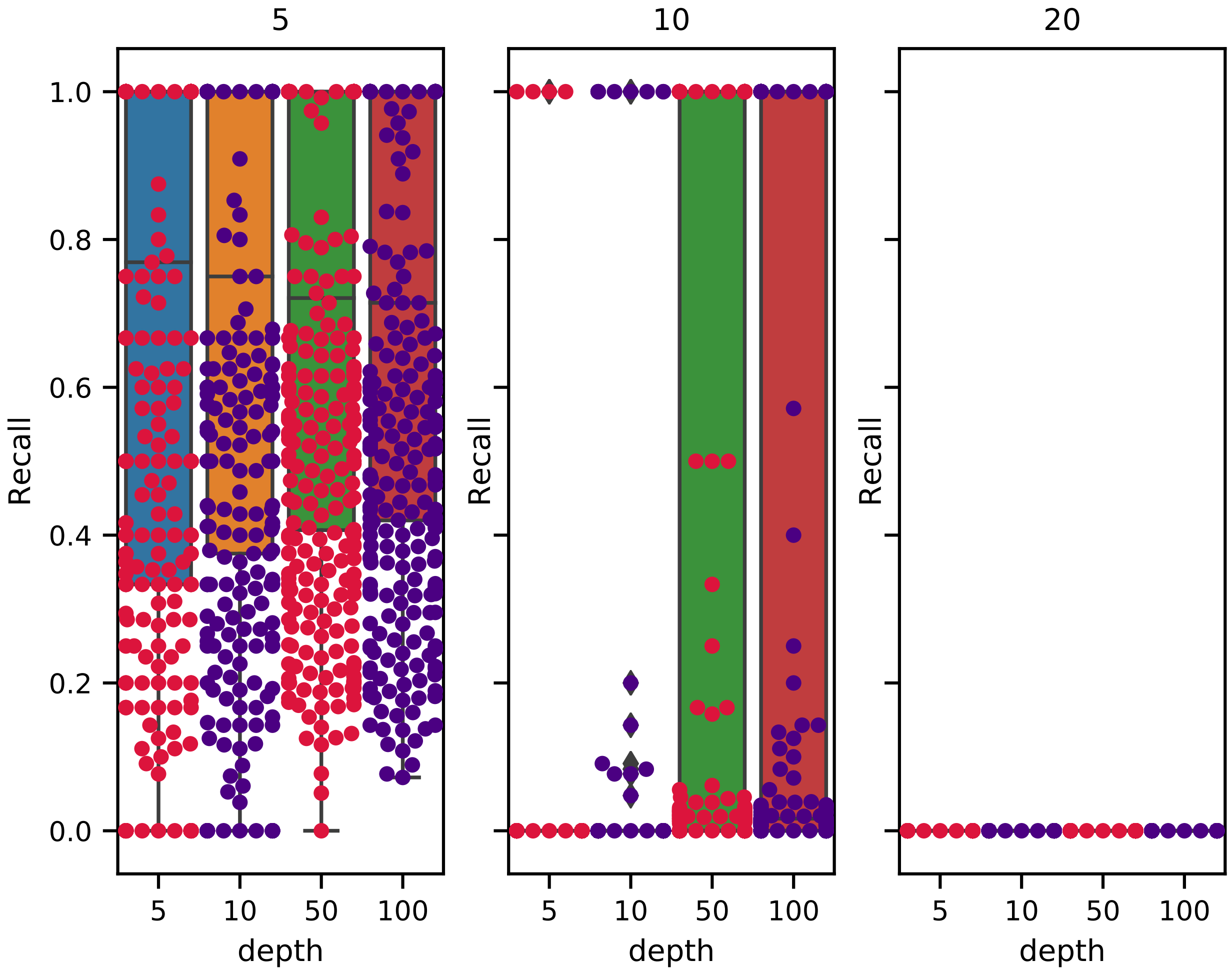

Figure 66: **Low quality precision per construct similarity 90% .** This plot shows the precision related to the low quality setting, with parts similarity $= 90\%$. The precision is evaluated on each each Nanogate output, each point represent the precision evaluated on a single output. (The plot is correct, there is a bug with labels, this plot shows the precision)

5.8 Latest version V5.0.0

In this section I am showing for the first time results related to the parts order.

5.8.1 High quality reads analysis

Reads quality analysis The following plot shows the ratio between the reads who passed the Nanogate filter and the the reads marked as pass. The reads marked as pass are all the reads to which Nanogate assigned the correct number of parts, as consequence these reads are long enough to include a number of parts equal to the number of parts in the original construct.

$$\text{Parts analysed} = \text{pass reads}/\text{total reads} \tag{4}$$

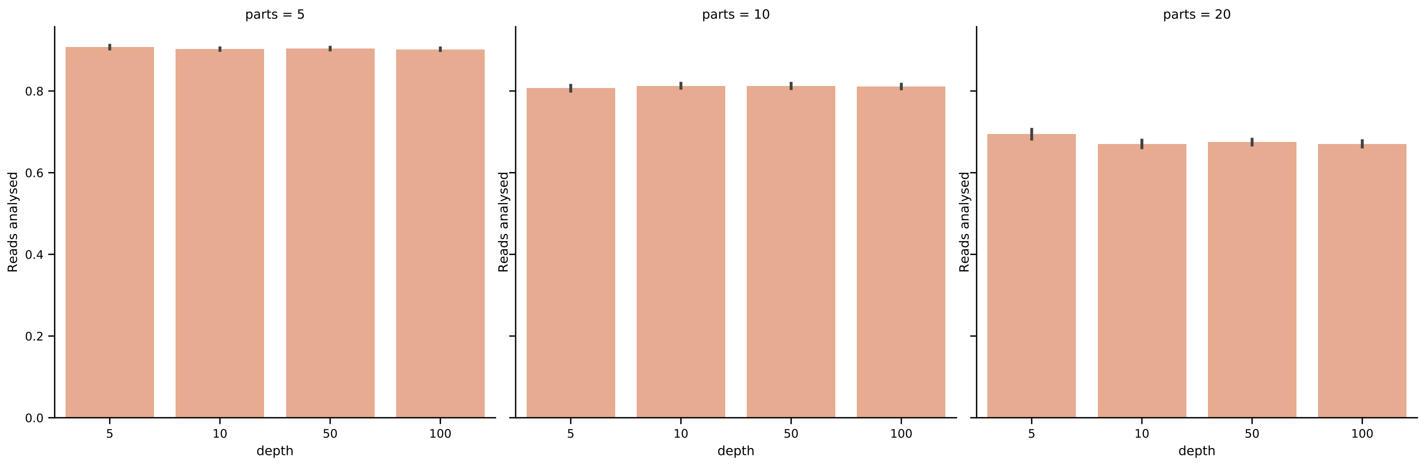

Figure 67: **High quality reads analysis.** This plot shows the percentage of reads analysed in relation to the depth and number of parts.The number of parts is the main factor affecting the number of reads analysed followed to the depth. The reads analysed are high quality and there is no similarity between parts.

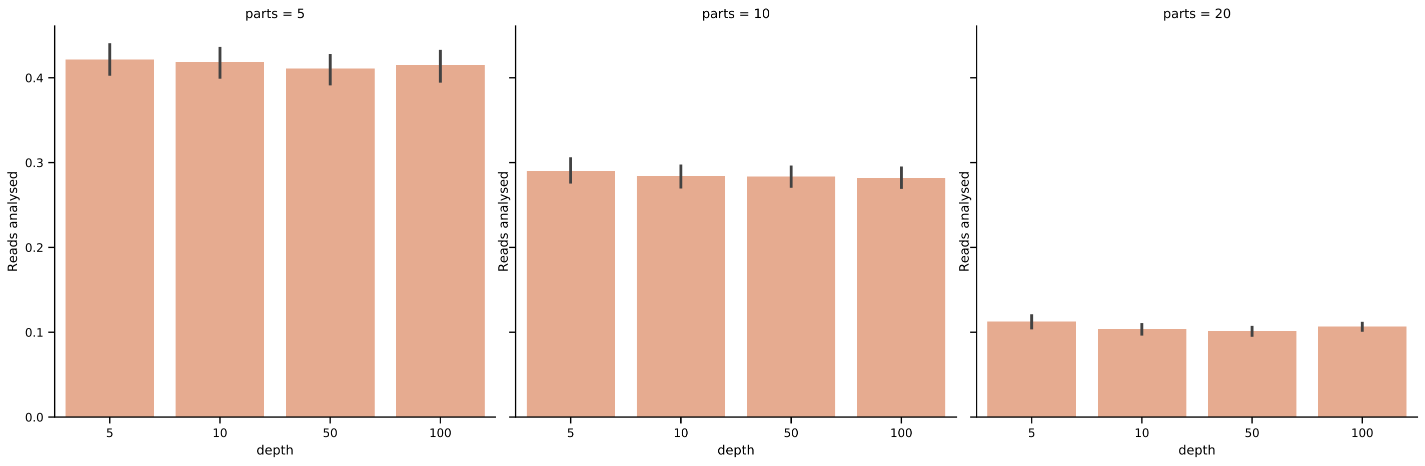

Figure 68: **Low quality reads analysis.** This plot shows the percentage of reads analysed in relation to the depth and number of parts.The number of parts is the main factor affecting the number of reads analysed followed to the depth. The reads analysed are low quality and there is no similarity between parts.For the low quality setting the percentage of the reads analysed is at maximum 40% for constructs with 5 parts, half of the high quality setting. The minimum is instead 10%.For the low quality setting the percentage decrease linearly with the the number of parts per construct.

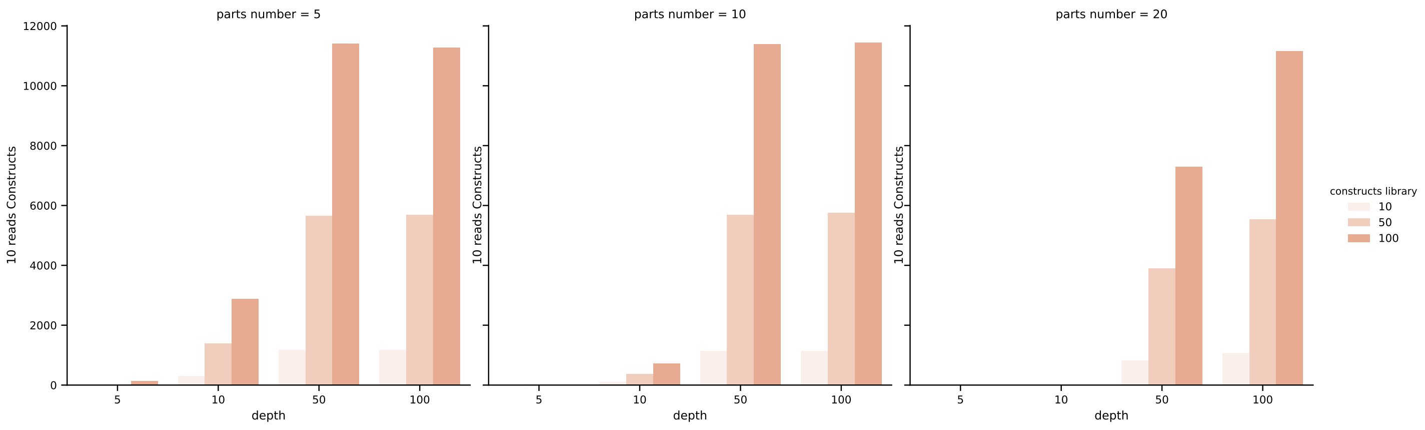

Figure 69: **High quality constructs with at least 10 reads.** Here I show the amount of constructs with more than 10 reads.In this plot is clear how the depths 5x and 10x generate a low amount of reads, in particular for 10 and 20 parts constructs there is almost no construct with more than 10 reads. The number of reads per constructs is affected by the depth, the size of the constructs library and the number of parts per constructs. The reads analysed are high quality and there is no similarity between parts

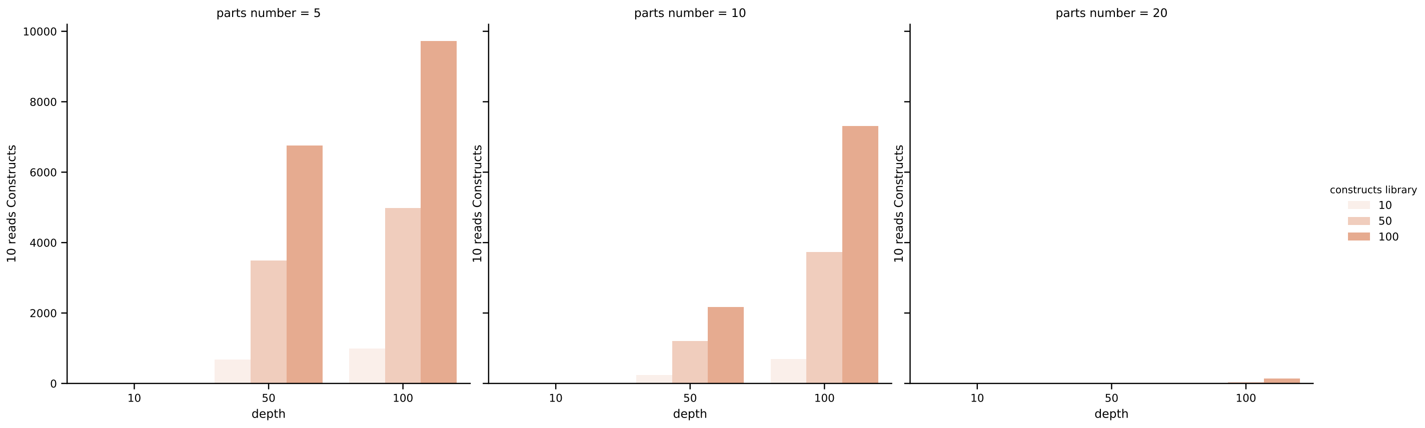

Figure 70: **Low quality constructs with at least 10 reads.** Here I show the amount of constructs with more than 10 reads.In this plot is clear how the depths 5x and 10x generate a low amount of reads. Indeed, for $5x$ and $10x$ depths there is no construct with more than 10 reads. For construct with 20 parts none of the depths generates constructs with more than 10 reads The number of reads per constructs is affected by the depth, the size of the constructs library and the number of parts per constructs. The reads analysed are low quality and there is no similarity between parts

5.8.2 Nanogate qualitative analysis

Here I show plots related to the qualitative analysis of Nanogate taking into account results related only the parts present or absent in construct. I do not take into account the parts order. In particular I am analysing the precision

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

TP = Nanogate predicted correctly every part in the construct FP = Nanogate did not predict correctly every part in the construct and the composition of the construct match another construct in the library

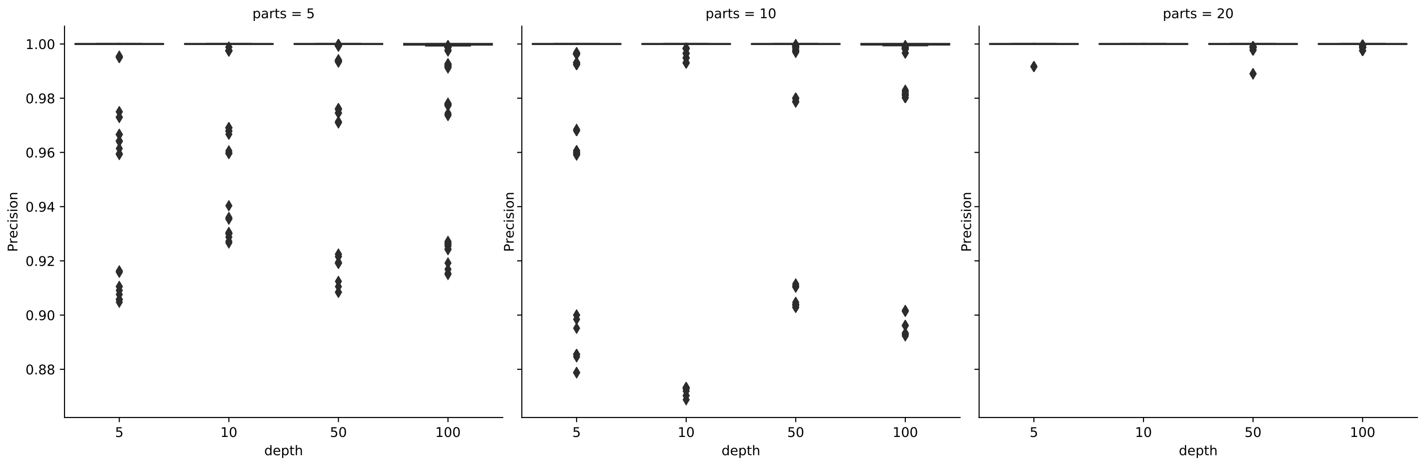

Figure 71: **High quality precision per result 0% similarity.** This plot shows the precision related to the high quality setting, with parts similarity $= 0\%$. The precision is evaluated on each Nanogate output, each point represent a single Nanogate result table. The precision is almost 100% with few out liars decreasing with the number of parts.

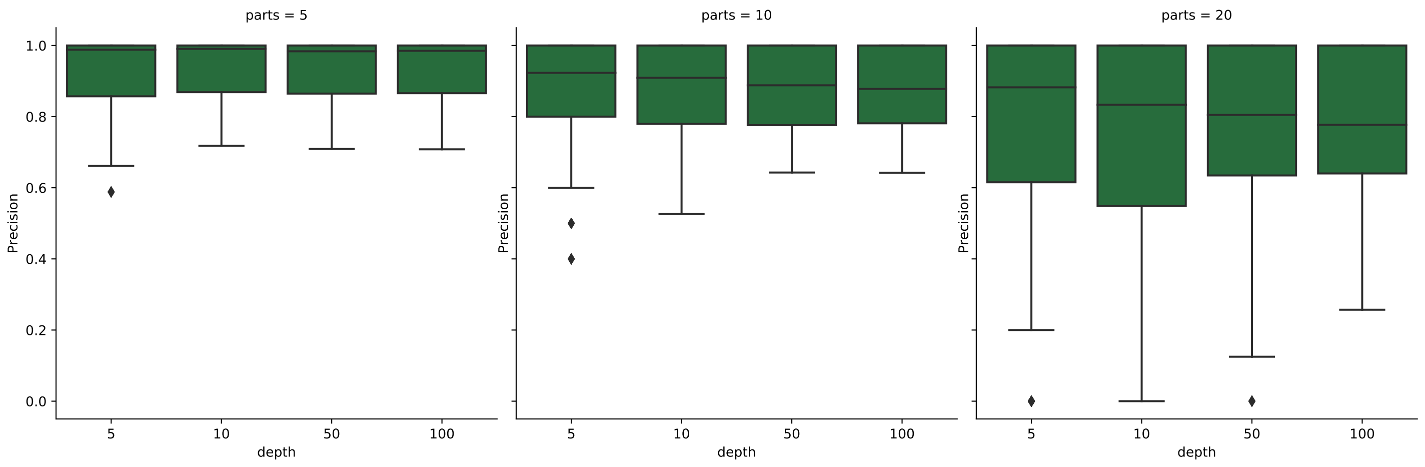

Figure 72: **Low quality precision per result 0% similarity.** This plot shows the precision related to the low quality setting, with parts similarity $= 0\%$. The precision is evaluated on each each Nanogate output, each point represent a single Nanogate result table. The precision decrease sensibly with the number of parts per construct, although being on average 90%, the variance sensibly decrease by increasing the number of parts per construct and decreasing the depth.

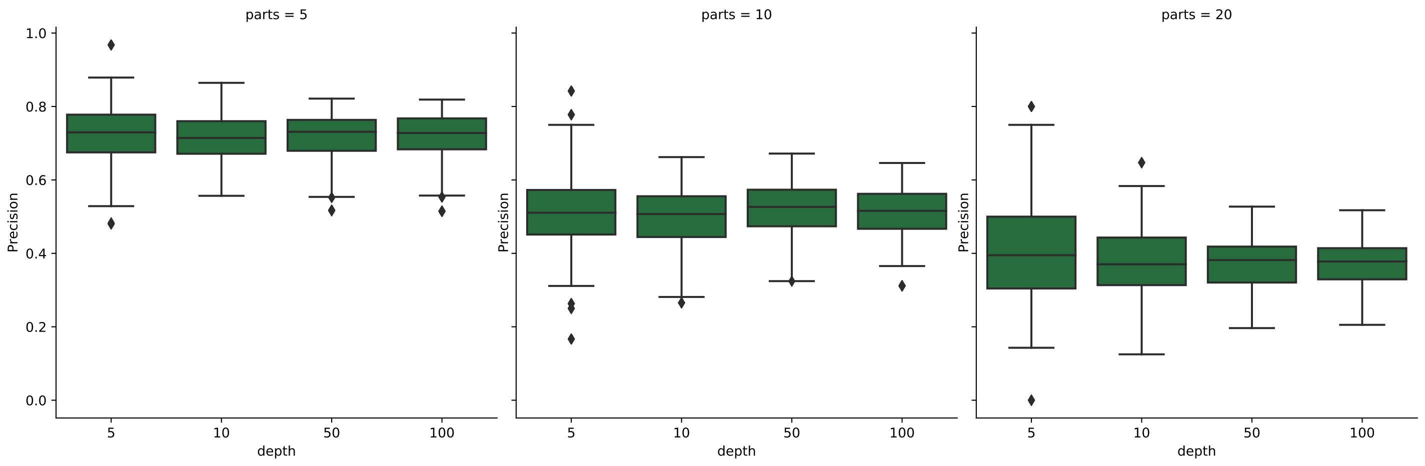

Figure 73: **High quality precision per result 90% similarity.** This plot shows the precision related to the high quality setting, with parts similarity $= 90\%$. The precision is evaluated on each Nanogate output, each point represent a single Nanogate result table. Here the precision is on average 50%, this plot prove that the similarity between parts affects the precision more than quality of the reads. The precision decrease sensibly with the number of parts per construct, the variance sensibly decrease by increasing the number of parts per construct and decreasing the depth

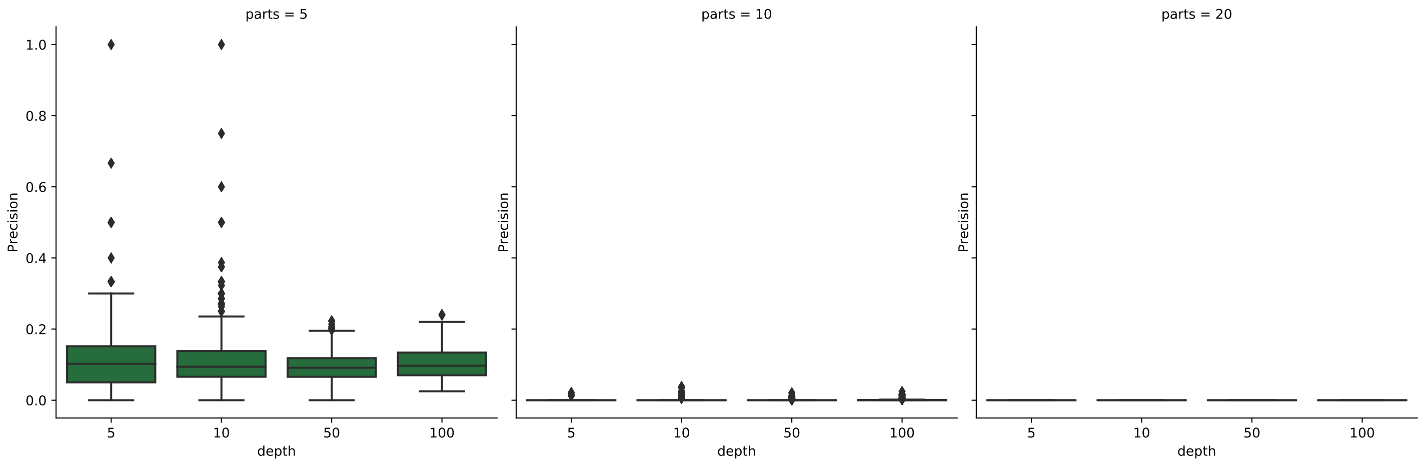

Figure 74: **Low quality precision per result 90% similarity.** This plot shows the precision related to the high quality setting, with parts similarity $= 90\%$. The precision is evaluated on each Nanogate output, each point represent a single Nanogate result table. Here the precision is always below 20%. This plot shows Nanogate performances in the worst case scenario, where the similarity of parts is 90% and the quality of reads is low.

5.8.3 Nanogate parts order analysis

Here I take into account two different parameters the parts order ratio and the parts order precision.

Parts ratio = Parts order true positives / true positives

Here I measure how many of the reads analysed correctly by Nanogate (Nanogate assigned the same parts included in the original construct) are ordered correctly by Nanogate.

The second measure, the precision,is independent from the Nanogate qualitative analysis:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

TP = Nanogate predict the exact order of parts in a construct FP = Nanogate did not predict correctly the order of every part in the construct and the composition predicted by Nanogate matches another construct in the library

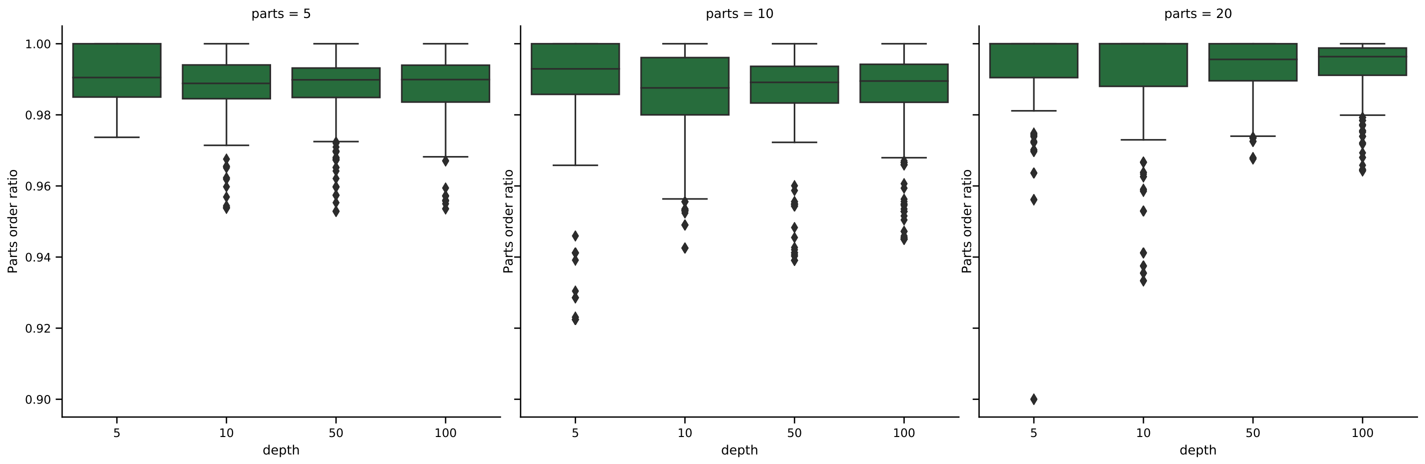

Figure 75: **High quality Parts order ratio 0% similarity.** This plot shows the parts ratio related to the high quality setting, with parts similarity $= 0\%$. The parts ratio is evaluated on each Nanogate output, each point represent a single Nanogate result table. Here the parts order is on average 99%, and stable across the different parameters. This plot shows that for the high quality setting and in absence of parts similarity, 99% of the reads to which Nanogate assigned the correct parts are ordered correctly.

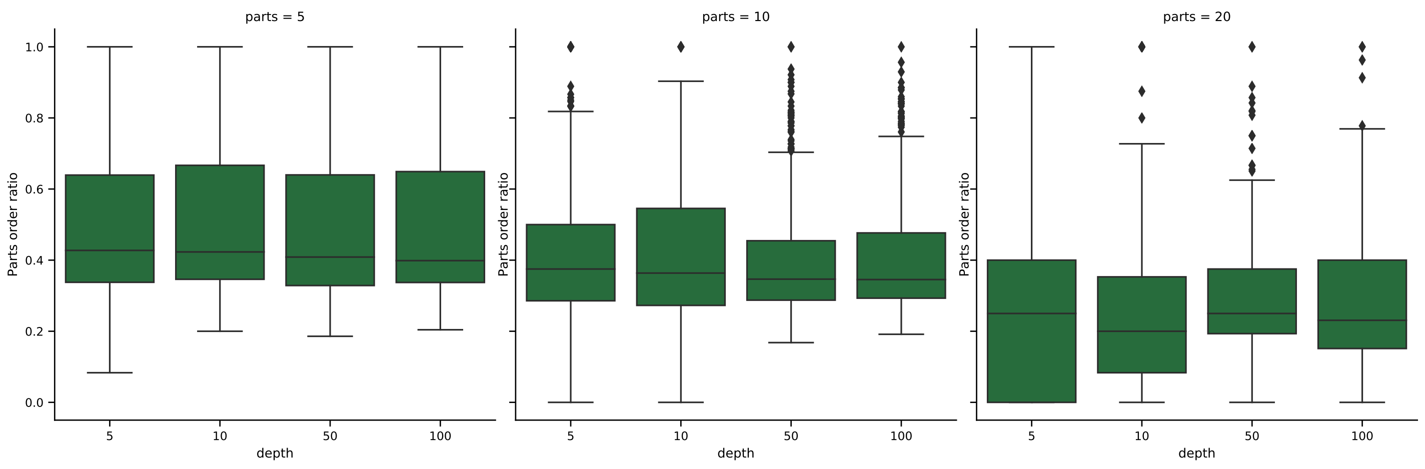

Figure 76: **Low quality Parts order ratio 0% similarity.** This plot shows the parts order ratio related to the low quality setting, with parts similarity $= 0\%$. The parts order ratio is evaluated on each Nanogate output, each point represent a single Nanogate result table. Here the parts order is on average 40%. This plot shows how the quality of the reads affect the parts order algorithm, if comppared with the previous plot

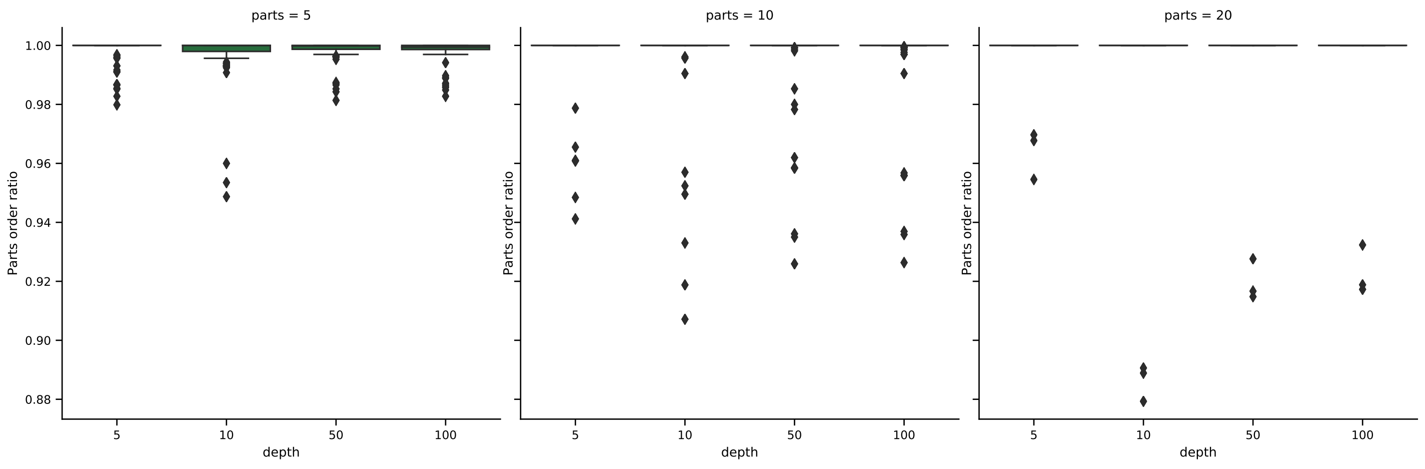

Figure 77: **High quality Parts order ratio 90% similarity.** This plot shows the parts order ratio related to the high quality setting, with parts similarity $= 90\%$. The parts order ratio is evaluated on each Nanogate output, each point represent a single Nanogate result table. Here the parts order is on average 99%. This plot shows how the similarity between reads does not affect the parts order algorithm. The results in this plot is comparable to the results for high quality and 0% similarity

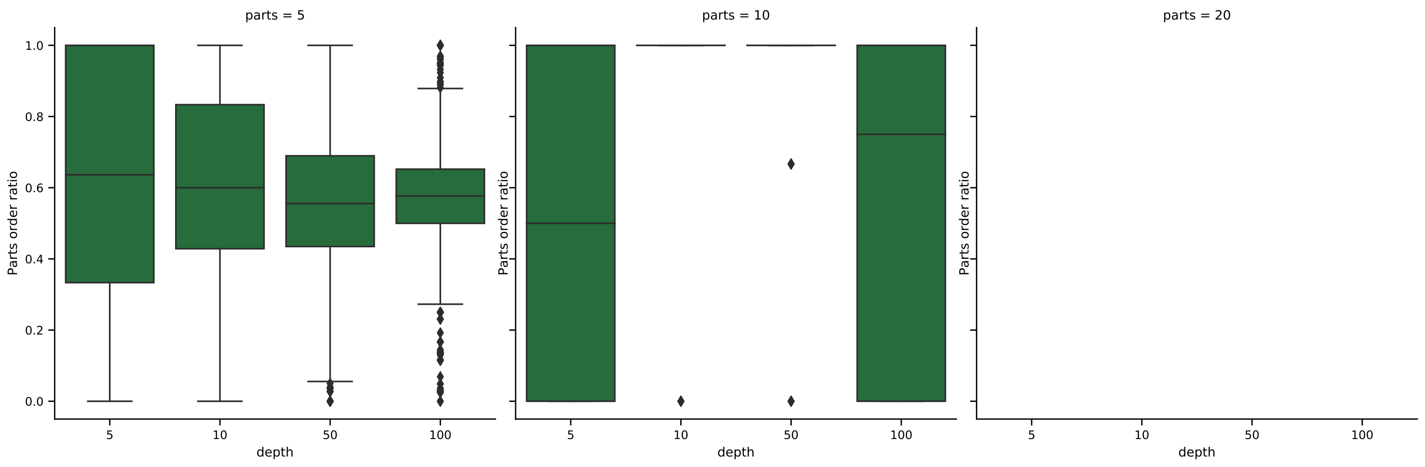

Figure 78: **Low quality Parts order ratio 90% similarity.** This plot shows the parts order ratio related to the low quality setting, with parts similarity $= 90\%$. The parts order ratio is evaluated on each Nanogate output, each point represent a single Nanogate result table.The results of the plot are consistent with the low quality 0% similarity, if we observe the 5 parts construct. Due to the quality of the reads as described in fig 68 and fig 70 the results for 10 and 20 parts are not reliable, due to the low amount of reads analysed.

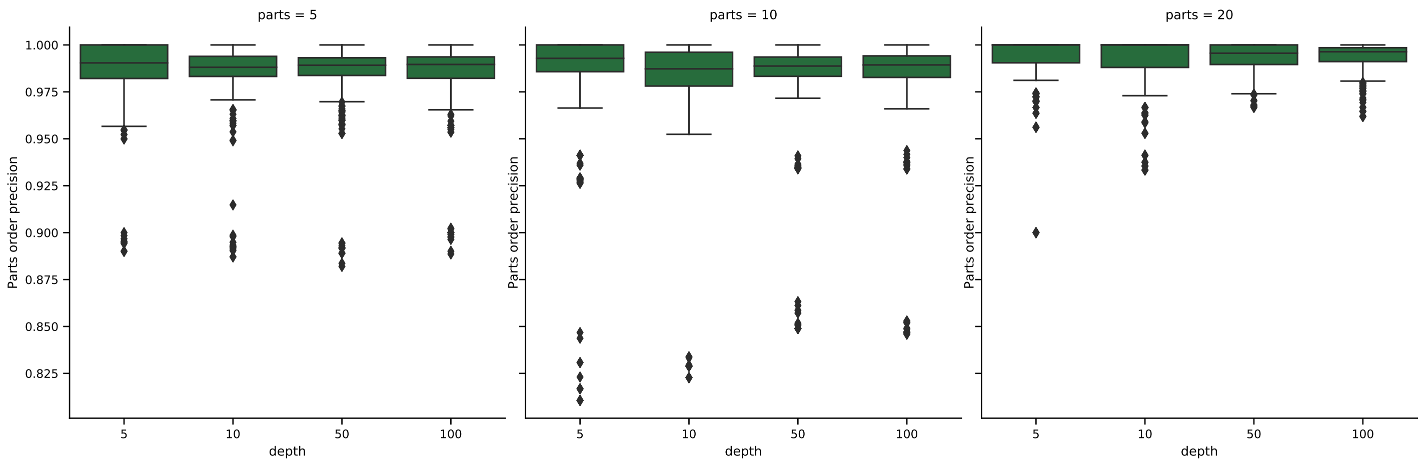

Figure 79: **High quality Parts order precision 0% similarity.** This plot shows parts order precision related to the high quality setting, with parts similarity $= 0\%$. The parts order precision is evaluated on each Nanogate output, each point represent a single Nanogate result table. These results are consistent with those in fig 71. Since the parts order ratio in fig 75 is 99% on average, the results of this plot and fig 71 are almost identical.

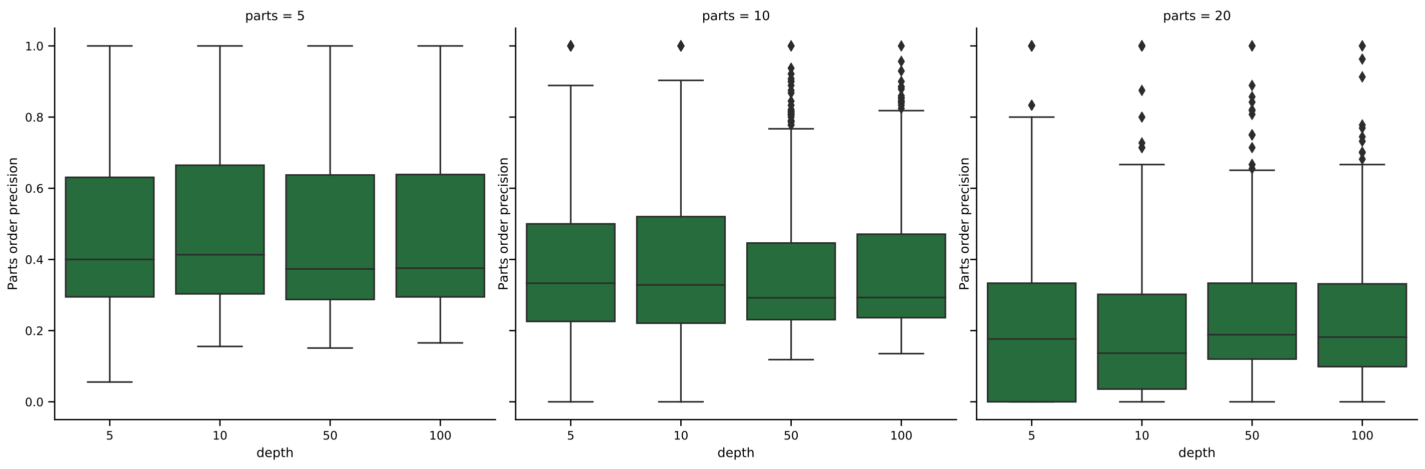

Figure 80: **Low quality Parts order precision 0% similarity.** This plot shows parts order precision related to the low quality setting, with parts similarity $= 0\%$. The parts order precision is evaluated on each Nanogate output, each point represent a single Nanogate result table. These results are consistent with those in fig 72 and fig 76.

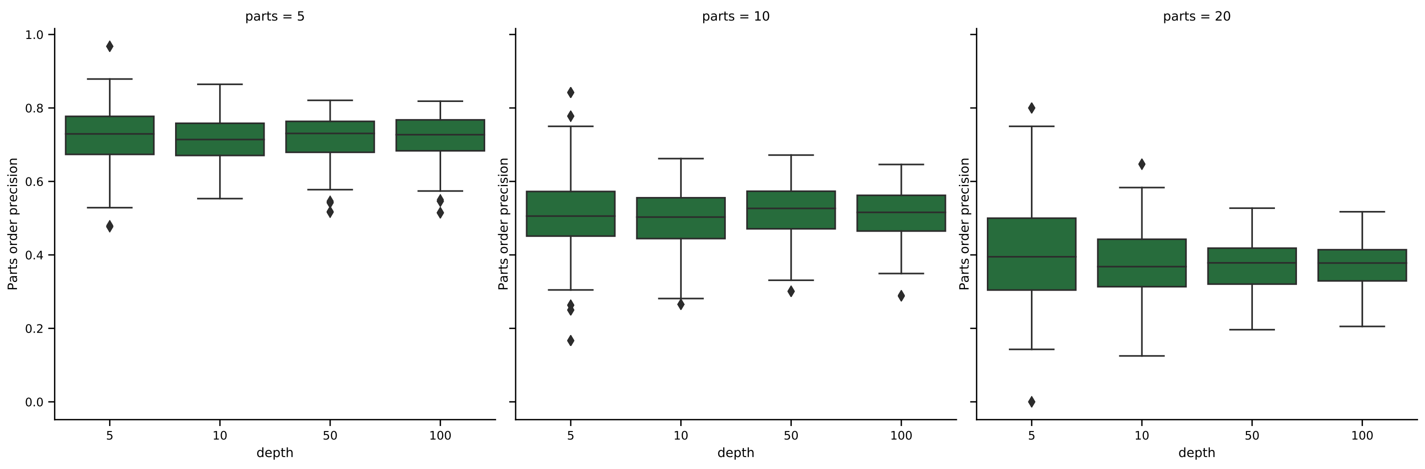

Figure 81: **High quality Parts order precision 90% similarity.**This plot shows parts order precision related to the high quality setting, with parts similarity $= 90\%$. The parts order precision is evaluated on each Nanogate output, each point represent a single Nanogate result table. These results are consistent with those in fig 73. Since the parts order ratio in fig 77 is 99% on average, the results of this plot and fig 73 are almost identical.

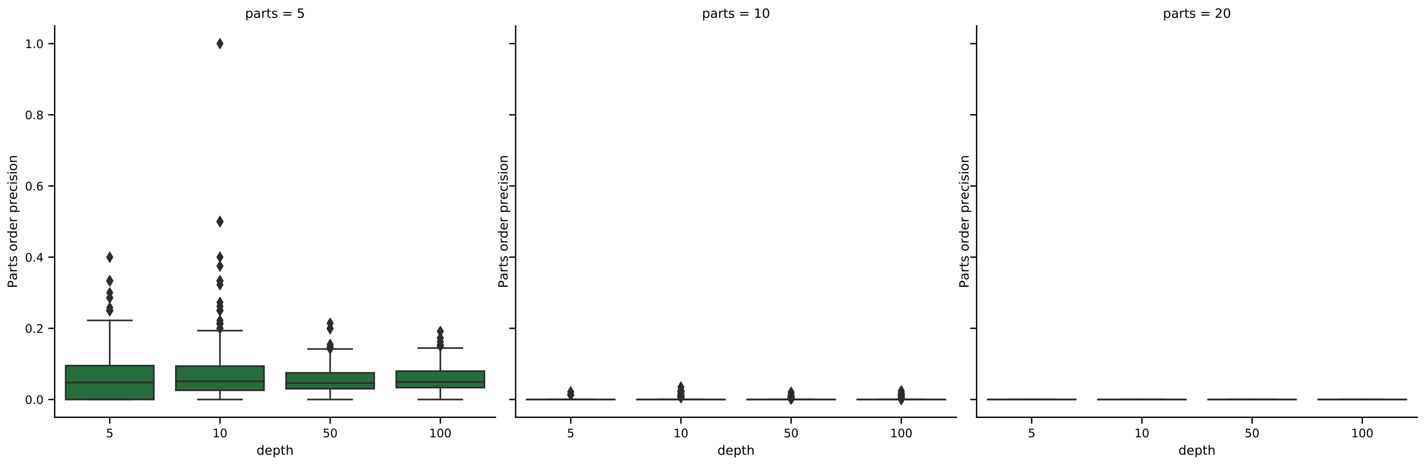

Figure 82: **Low quality Parts order precision 90% similarity.** This plot shows parts order precision related to the low quality setting, with parts similarity $= 90\%$. The parts order precision is evaluated on each Nanogate output, each point represent a single Nanogate result table. These results are consistent with those in fig 78 and fig 74. The parts order ratio in fig 78 is 60% for 5 parts construct,and the precision for the parts order for 5 parts constructs is lower than the standard precision. Results for 10 and 20 parts constructs are not reliable due to the low amount of reads, see fig. 68 and fig. 70
.

5.8.4 Contaminants analysis

Here, I analyse how Nanogate detects contaminants. In particular I consider and mark as contaminants chimeras, junk and random reads. All the reads belonging to this category are analysed separately. In the contaminants analysis I ignore the chimeras, and count as a contaminant detected correctly only junk and random reads to which Nanogate assigned no parts.

$$\text{Contaminant discovery} = \text{Contaminants with no part assigned all junk and random reads} \tag{7}$$

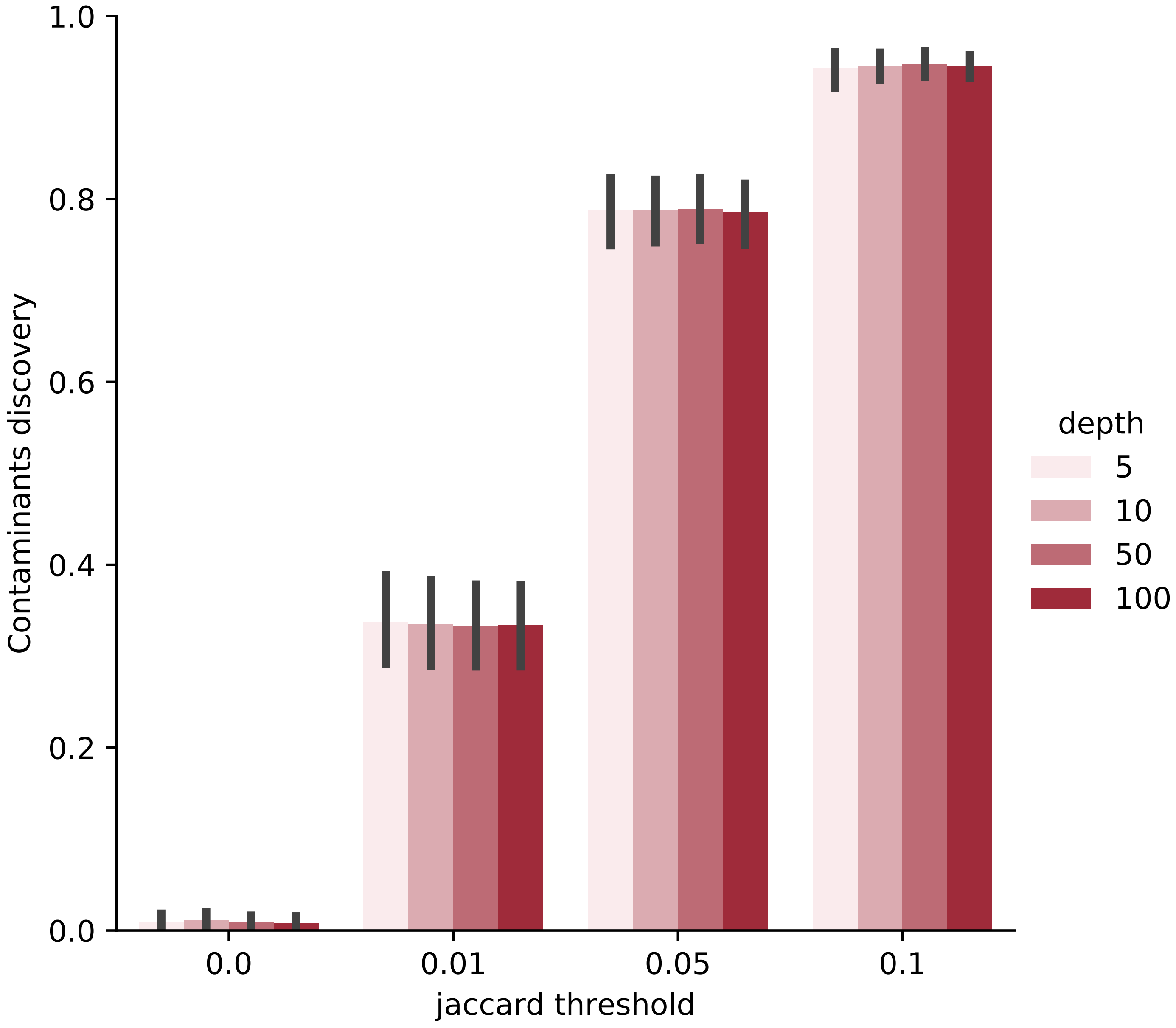

Figure 83: **Contaminant discovery..**This plot is realted to the low quality setting with parts similarity = 0%. Here I show how increasing the jaccard threshold the contaminant discovery increases as well. In particular Jt 0.05 and 0.1 have a discovery rate between 80% and 90%. The plot in fig.72 shows how the thresholds allow a precision up to 80%, and a contaminant discovery up to 90%
.