# Genome Assembly Using Overlap Graphs

**236522 – Algorithms in Computational Biology, Alternative Assessment**
**Roi Teichman | 318865524 | roi.teichman@campus.technion.ac.il**

**Abstract**

This project evaluates an overlap graph-based genome assembly pipeline using simulated next-generation sequencing reads with controlled parameters. We systematically investigated how the number of reads ($N$), read length ($l$), sequencing error probability ($p$), and $k$-mer filtering threshold ($k$) affect assembly quality across coverage scenarios ranging from sub-$1x$ to $30x$. Our implementation employs a modified global alignment for overlap detection, followed by a greedy cycle removal and topological sorting strategy for contig assembly. Key findings include: (1) N50 values typically approximate read length, with exceptions at higher read counts or when k-mer filtering is disabled; (2) increasing coverage does not consistently improve assembly quality, as higher coverage often leads to increased mismatch rates and more, shorter contigs; and (3) unfiltered assembly produces superior results at lower coverage but becomes computationally prohibitive at higher coverage. Optimal performance was observed with moderate error rates ($p \leq 0.01$) and intermediate $k$ values ($k = 5$). These results underscore the complex interplay between sequencing parameters and assembly outcomes, demonstrating that overlap-graph approaches remain viable for genome assembly when parameters are carefully optimized.

## Introduction

Genome assembly is a fundamental problem in computational biology, aiming to reconstruct an organism's genome from short sequencing reads. Next-generation sequencing (NGS) technologies, such as those developed by Illumina, typically produce short reads—often in the range of $50\ to\ 150\ base\ pairs\ (bp)$—with raw error rates between 0.001 and 0.1. These technologies have significantly improved the throughput and cost-effectiveness of sequencing but present computational challenges due to short read lengths and sequencing errors (Shendure & Ji, 2008)[1].

Among the various approaches for genome assembly, overlap-based methods reconstruct sequences by identifying pairwise overlaps between reads and constructing an overlap graph (Myers, 2005)[2]. A classical approach for genome assembly is based on constructing an overlap graph, where nodes represent reads and edges indicate significant overlaps between them. Pioneering work by Lander and Waterman (1988)[3] provided a mathematical foundation for understanding genome coverage and the probabilistic nature of read overlaps.

In modern contexts, assembly algorithms often use variations of the Needleman-Wunsch algorithm for Global Alignment[4], adapted for overlap detection, to determine the optimal way to merge reads. In parallel, local alignment methods are used to assess the quality of contigs and estimate error rates. Unlike de Bruijn graph approaches, which break reads into fixed-length $k$-mers, overlap graphs preserve longer contiguous sequences and are particularly beneficial when working with moderate-length reads (Simpson & Durbin, 2012)[5]. However, constructing and processing overlap graphs efficiently requires optimized methods for overlap detection and graph simplification.

In this study, we implemented a genome assembly pipeline based on overlap graphs and systematically evaluated the impact of key parameters—number of reads ($N$), read length ($l$), sequencing error rate ($p$), and optimization parameter of minimal overlap length ($k$, from $k$-mers variation)—on assembly

performance. Inspired by classical models of genome coverage (Lander & Waterman, 1988), we designed experiments to explore parameter variations systematically and optimize assembly performance. We aim to characterize the relationship between these parameters and standard assembly metrics, including the number of contigs, N50, genome coverage, and mismatch rate (of the aligned region and of the whole genome).

## Methods

### Read Simulation

To evaluate the performance of our assembly approach, we simulated sequencing reads from a reference genome of $Phi\ X\ 174$ (approximately $5{,}400\ bp$). For a given number of reads ($N$) and read length ($l$), we generated reads by sampling contiguous substrings from the genome with uniform coverage. We introduced substitutions at a specified error probability ($p$) to model sequencing errors, simulating real-world sequencing artifacts. Although many bacterial and viral genomes are circular, we treated the genome as linear, reflecting the format in which FASTA files store sequences. Furthermore, circular genomes present additional computational challenges during assembly, as described by J. et al. (2005)[6] and by Shendure and Ji (2008), particularly when resolving cycles in the overlap graph.

### Overlap Graph Construction with $k$-mer Optimization

An overlap graph is a structure where nodes represent reads, and directed edges between nodes represent overlaps between sequences (specifically, a suffix of the first sequence overlapping with a prefix of the second). The weight of each edge is determined by the method used to identify overlaps, making the overlap identification process crucial.

A naive approach would be to check the overlap length between every pair of reads, which would require $O(N^2 \times l)$ time complexity (comparing all pairs of reads and examining potential overlaps). However, this method fails to account for sequencing errors within overlap regions, potentially underestimating true overlap lengths.

Alternatively, aligning all pairs using a variation of the Needleman-Wunsch Global Alignment algorithm provides higher confidence in overlap detection and can accommodate errors within overlap regions (Xia, 2007[7]). This approach, however, has a computational complexity of $O(N^2 \times l^2)$, which becomes prohibitive for large datasets. In our experiments, we observed that for $N > 10{,}000\ bp$, both graph construction and subsequent cycle removal became computationally intensive.

To balance accuracy and computational efficiency, we implemented a $k$-mer-based filtering strategy (conceptually rooted in the work of Claude Shannon, 1948[8]). Jenike et al. (2025)[9] described common applications of $k$-mers, including comparing sequencing libraries. Based on this insight, we developed the following algorithm:

1. We constructed a $k$-mer index from all read prefixes.
2. For each read, we checked whether its suffix had a corresponding prefix in the $k$-mers index.
3. Only if a match was found and it was not from the same read, we performed full overlap alignment.

This optimization relies on the assumption that during contig assembly, edges with higher weights (representing better overlaps) will be prioritized. Therefore, we can focus on including edges with the highest potential to be chosen for any node, reducing the number of edges representing weak

overlaps. The probability of encountering a sequencing error within a $k$-length sequence is $(1-p)^k$, so we aimed to select a $k$ value that significantly reduced candidate alignments while maintaining high performance.

To assess the overlap between each pair, we used an Overlap Alignment method, a variation of global alignment that does not penalize overhanging ends, with the following scoring scheme:

1. $match$ $+10$
2. $mismatch$ $-1$
3. $indel\ (gap)$ $-\infty$

The ratio between match and mismatch scores was chosen to reflect the maximum probability of sequencing errors ($p = 0.1$), hence the $1:10$ ratio. We assigned an infinite penalty to indels because our error model only introduces substitutions, not insertions or deletions. Optimization of these parameters could be addressed in future work.

**Cycle Removal, Topological Sort and Contig Assembly**

After detecting significant overlaps, we constructed a directed overlap graph where nodes represent reads and edges indicate overlaps. Due to sequencing errors and repetitive regions, the graph often contains cycles, which must be resolved before linearizing the genome assembly. Inspired by the "Linear Transitive Reduction Algorithm" of Myers (2005), we implemented a greedy edge removal strategy:

1. While cycles exist, identify the weakest overlap (edge with lowest alignment score).
2. Remove the weakest edge and repeat until the graph is acyclic.
3. Perform topological sorting to order reads along the genome.
4. Traverse from the first node to assemble the fewest and longest contigs possible.

We chose to remove the weakest edge in any cycle because it is likely the one that would not be selected in a natural assembly process. This approach could be further refined by protecting edges that are the only connection for a node, but we leave this improvement for future work. We employed topological sorting to increase the likelihood of assembling fewer and longer contigs. This is important because each read can be visited only once, so starting with a read from the middle of the genome might prevent connecting to contigs from the beginning of the genome if their nodes have already been visited.

With the directed acyclic graph (DAG) obtained from cycle removal, we assembled contigs by traversing paths according to the topological sort order. Contig sequences were constructed by merging reads based on their overlap relationships. This approach ensures a feasible assembly path while mitigating errors from spurious overlaps.

**Quality Assessment**

To evaluate assembly quality, we computed the following metrics:

1. Number of Contigs: Lower values indicate better assembly continuity.
2. Genome Coverage (%): Fraction of the reference genome covered by the assembled contigs.
3. N50: The length of the shortest contig at 50% of the total assembly length, representing the median of the distribution by mass.

4. Mismatch Rate (Aligned Regions): Proportion of mismatches in regions where contigs align to the reference.

5. Mismatch Rate (Genome Level): Similar to the above but also penalizing uncovered bases.

To compute coverage and mismatch rates, we used the Smith-Waterman local alignment algorithm[10] to align contigs with the reference genome. For that, we use the same scoring function except that we change the score for indle from $-\infty$ to -1 (as the mismatch gets), because here they may occur. Using the local alignment like that allowed us to identify covered regions and determine the genome depth (number of covering contigs) for each position. By analyzing these alignments base by base, we quantified the mismatch rates.

**Parameter Exploration**

To systematically assess the impact of $N, l, p$, and $k$, we designed the following controlled experiments:

1. Varying number of reads ($N$): Fixed read length while varying error probabilities and k parameter to observe effects on assembly metrics.

2. Varying read length ($l$): Fixed $N$ and examined the influence of different error probabilities and $k$-mer parameter values on assembly metrics.

3. Varying l and N together: For a genome of length $G$, and $N$ reads of length $l$, the average (expected) coverage is calculated by: $C = \frac{N \times l}{G}$. In this experiment, for fixed $C$, we increased read length ($l$) and decreased proportionally the number of reads ($N$) to maintain constant coverage, examining the influence of different combinations of $p$ and $k$ on assembly metrics.

The experiments were conducted on an Intel architecture using multiprocessing to enhance resource utilization, and each experimental condition was run for 10 iterations. We explored the following parameter ranges:

1. $N$: Five values sampled logarithmically between 100 and 10,000: {100, 316, 1000, 3162, 10000}[i].

2. $l$: Three values sampled linearly between 50 and 150: {50, 100, 150}.

3. $p$: Three values sampled logarithmically between 0.001 and 0.1: {0.001, 0.01, 0.1}.

4. $k$: Four values sampled linearly: {0, 5, 10, 15}[ii].

5. $C$: five values sampled based on interesting cases: {0.92[iii], 2, 5, 10, 30}.

The chosen coverage values represent critical scenarios in genome assembly. For example, when $C < 1$, the genome is covered only a few times, while higher $C$ values correspond to near-complete coverage. According to Sims et al. (2014), sequencing depths in the range of 20x–30x are generally considered sufficient for achieving near-complete genome coverage and reliable variant detection in genomic analyses. Therefore, we included 30x as a high-coverage scenario, as it represents a balance between data quality and diminishing returns from increased sequencing depth; further increases beyond 30x typically yield only marginal improvements at significantly higher computational costs.

---

[i] Increased N trials resulted in stack overflow. Multiprocessing, though a potential factor, is required for timely results.

[ii] The k=0 experiment completed for C=0.92, 2, and 5. For C=10 and 30, edge removal was incomplete after 48 hours, attributed to graph complexity. So, for C=10 and C=30, there are just 3 values of k.

[iii] This is the minimum coverage that can be obtained by $N = 100$ and $l = 50$, by the formula.

Given that the average (expected) coverage is calculated by the formula $C = \frac{N \times l}{G}$, where $G$ is constant, the experiments naturally fall into two categories:

1. Varying Coverage (Experiments 1 and 2): In these experiments, either the number of reads or the read length was increased while keeping the other parameter constant, leading to a proportional increase in genome coverage.
2. Constant Coverage (Experiment 3): In this scenario, the genome coverage was maintained at a fixed level by increasing the read length and proportionally decreasing the number of reads. This allowed us to isolate the impact of read length on assembly quality while holding overall coverage constant.

These experiments provide insights into the optimal conditions for genome assembly and the robustness of the overlap-graph approach under varying sequencing parameters.

**Code**

All implementations and experiments are available in the following GitHub repository: https://github.com/roiteichman/Genome-Assembly-Using-Overlap-Graphs

## Results

### Experiment 1 — Varying Number of Reads

We examined the influence of varying the number of reads on assembly metrics. We hypothesized that increasing $N$ would lead to improved results, specifically fewer but longer contigs (decreased number of contigs and increased N50), lower mismatch rates, and increased genome coverage.

In practice, for all read lengths ($l$ = 50, 100, 150), as $N$ increased, we observed longer contigs with better genome coverage. The genome coverage results matched our expectations, with coverage increasing as $N$ increased. Additionally, we observed that for the same $N$, genome coverage increased with read length $l$ (Appendix 1). An additional valuable observation was the visualization of genome coverage depth (number of copies covering each base in the genome) and genome coverage rate (indicating whether each base was covered by any contig) for each iteration. For example, examining the coverage rate and coverage depth for $N = 100,\ l = 50$ with $p = 0.001,\ k = 5,$ , we observe that not all bases are covered as shown in Appendix_1_A. This is expected since the calculation by the formula is: $C_{\{N=100,l=50\}} \approx \frac{5000}{\sim 5400} \approx 0.92 < 1$. Conversely, for larger $N$ and $l$, we anticipate complete genome coverage with depth corresponding to expected coverage. For instance, with $N = 10000, l = 150, p = 0.1, k = 5$, the genome coverage approaches 100% with only occasional uncovered bases, as illustrated in Appendix_1_B.

For each fixed $l$, we found that the average N50 value across all combinations of $p$ and $k$ was exactly equal to $l$ (as shown by the trend line in Appendix_2). This indicates that $l$ was the median contig length, suggesting that most contigs were simply original reads that did not connect to other reads to form longer assemblies.

Interestingly, for $l = 100$ and $150$, only configurations with $k = 5$ achieved $N50$ values higher than the average $N50 = l$, regardless of p value. Within this subset, $N50$ increased with $p$. This suggests that for these read lengths, higher N50 values might result from errors in reads creating overlaps, while larger k values may restrict overlap connections and impede assembly. However, for $l = 50$, the two highest $N50$ values were achieved with $k = 15$ and p = 0.001 or 0.01 (Appendix_3). This was unexpected, as requiring an overlap of at least $30\%$ of the read length would create a sparse overlap graph with fewer edges, yet these configurations performed exceptionally well.

Two surprising results emerged regarding the number of contigs and mismatch rates. Contrary to our hypothesis that increasing $N$ would decrease the number of contigs and improve contig quality, we observed that as $N$ increased, both the number of contigs (Appendix_4) and mismatch rates (Appendix_5) increased. For the number of contigs, there were minimal differences between different $l$ values.

These results, combined with our N50 findings, suggest that most contigs are simply individual reads without connections to other reads, explaining the high number of contigs and their short median length (N50) equalling $l$. Furthermore, the general trend of increasing mismatch rates (both for aligned regions and the whole genome) with increasing $N$ contradicted our expectation that more reads would provide better coverage for correct assembly. Interestingly, for all $l$ values, we did observe the expected decrease in mismatch rate for the whole genome (penalizing uncovered bases) between $N = 100$ and $N = 316$. However, this trend did not continue with further increases in $N$.

These results indicate that our algorithm faces challenges in assembling contigs as expected.

**Experiment 2 — Varying Read length**

Next, we examined the influence of varying read length on assembly metrics. Given the formula $C = \frac{N \times l}{G}$, we hypothesized that increasing l would yield improved results similar to our hypothesis for varying $N$: fewer but longer contigs, lower mismatch rates, and increased genome coverage.

As observed in the previous experiment, genome coverage improved with increasing l and N values, as demonstrated in Appendix_6. Similar to our earlier findings, in most cases $N50 = l$, indicating that our assembly process struggled to construct contigs longer than the individual reads themselves. This is evident from the alignment of values with $N50 = f(l) = l$. For $N = 3162$ and $N = 10000$, most parameter combinations achieved $N50 \approx l + 10$, suggesting that for $3162 \leq N$, our algorithm successfully constructed contigs beyond simply using the reads themselves. Exceptions occurred primarily with $k = 15$: first, with $p = 0.1, k = 15$, likely due to the large k value eliminating most edges and reverting the assembly to returning reads as contigs; second, with $p = 0.01, k = 15$, $N = 10000$, where $l = 50$ yielded $N50 \approx 85 > 50$, but $l = 100, 150$, behaved similarly to the first exception. The same pattern occurred with $p = 0.001, k = 15, N = 10000$, achieving $N50 \approx 150 > 50$ for $l = 50$ before reverting to the exception pattern. These results suggest that as $N$ increases beyond a certain threshold, our algorithm begins to assemble contigs as expected with appropriate $k$, values, as shown in Appendix_7.

As noted in the previous experiment (Appendix_4), the number of contigs is significantly affected by variations in N, while l has minimal impact, as reflected in Appendix_8, where contig numbers remain relatively constant across different settings (represented by different colors in each plot). This was unexpected as we anticipated that longer reads would increase assembly probability, thereby reducing contig numbers. Given that our initial assumptions did not materialize, this finding is particularly intriguing.

We discovered several interesting patterns regarding mismatch rates. First, for small $N$ values with specific parameters, we achieved our expected outcomes as mismatch rates decreased and remained low, indicating successful assembly of contigs that accurately represent the original genome. For example, with certain parameter combinations, we achieved total mismatch rates[iv] of approximately 0.1, signifying correct reconstruction of roughly 90% of the original genome, as shown in Appendix_9:

| $N$ | $l$ | $p$ | $k$ |
|---|---|---|---|
| 100 | 150 | 0.01, 0.001 | 5, 10, 15 |
| 316 | 50, 100 | 0.01, 0.001 | 10, 15 |
| 316 | 150 | 0.001 | 10, 15 |
| 1000 | 50 | 0.001 | 15 |

Second, we observed that parameter settings exhibit a layered organization, with $p = 0.001$ consistently yielding the lowest mismatch rates for each $N$ and $l$ combinations, while $p = 0.1$ produced the highest mismatch rates (with $k$ as a secondary factor, ordered from largest to smallest). Moreover, most settings displayed relatively flat gradients, indicating that variations in $l$ had less impact on mismatch rates compared to $N$. Finally, for $N \geq 1000$, mismatch rates generally increased with increasing $l$, suggesting that for high $N$ values, good mismatch rates cannot be achieved regardless of $l$.

[iv] Although the total mismatch rate penalized uncovered regions, leading to a potential bias, it nonetheless achieved satisfactory performance.

**Experiment 3 — Varying read length and the number of reads together**

Lastly, we examined the effect of simultaneously varying read length and number of reads while maintaining constant coverage. We hypothesized that increasing $l$ while decreasing $N$ would yield improved results, as longer reads increase overlap potential and should enhance performance.

Since we maintained constant coverage, genome coverage should theoretically approach 1 for any $1 \leq C$ remain consistent across different settings with the same C. This held for $2 < C$, while for $C = 0.92$ and $C = 2$, resulting in coverage rates below 1. This was primarily observed with $p = 0.1$, likely due to errors preventing contigs from fully covering the genome. However, cases with $p = 0.01$ and $k = 0$, also showed incomplete coverage, possibly because considering all edges as candidates led to the removal of "important" edges during our algorithm, resulting in contigs that did not cover the genome as expected. Apart from these exceptions, all other parameters achieved high coverage rates, as shown in Appendix_10.

Regarding N50, as previously observed, $N50 = l$, in most cases, indicating our algorithm's inability to assemble contigs longer than the reads themselves. This pattern persisted in this experiment, with exceptions for $C = 30$ and $k = 5$ which achieved $N50 \approx l + 10$ (as noted in Appendix_7), and for $C \in \{0.92, 2, 5\}$ with $k = 0$ which produced substantially longer N50 values (at least double the read length, and for $C = 5$, more than 10 times longer: $N50 > 10l$). These results suggest that our unoptimized algorithm without $k$-mer filtering could achieve significantly better performance and potentially superior assemblies. These findings are presented in Appendix_11.

For contig numbers, we finally observed our expected results: as $l$ increased (with N decreasing correspondingly), contig numbers decreased. Across all $C$ values, contig counts decreased by more than 50% from $l = 50$ to $l = 150$. Notably, contig numbers for $k = 0$ were at least 10 times lower than for any other $k$ across all $l$ values, further indicating the potential benefits of running our algorithm with $k = 0$. Additionally, lower $k$ values generally outperformed higher values; for example, with $C = 10$ and $C = 30$, $k = 5$ produced contig counts at least 50% lower than other k values. These findings are presented in Appendix_12.

Finally, regarding mismatch rates, we found that rates typically decreased with increasing read length for fixed C value, suggesting that longer reads yield better assemblies at constant coverage. Additionally, mismatch rates increased with higher C values when comparing the same $l$ values (similar to Appendix_9). Surprisingly, mismatch rates decreased with increasing $k$ value (at fixed $C$). This can be explained by the fact that higher $k$ value results in fewer candidate edges in the overlap graph; consequently, traversed edges are more likely to represent correct overlaps. Moreover, except for $C = 0.92$, the genome was fully covered by contigs, so no penalties were added. These results appear in Appendix_13.

Collectively, these results suggest that our method has significant potential when executed with optimal parameters ($N, l, p$, and $k$), potentially assembling fewer but longer contigs with mismatch rates indicative of high-quality assembly.

## Discussion

Our investigation into overlap-graph-based genome assembly reveals several key insights about the complex interplay between sequencing parameters and assembly quality. Through systematic exploration of read count ($N$), read length ($l$), error probability ($p$), and $k$-mer filtering threshold ($k$), we have identified both the strengths and limitations of this approach.

### Key Findings

#### The N50 Plateau Effect

One of the most consistent observations across our experiments was that N50 values typically approximated read length ($N50 \approx l$), indicating that our assembly algorithm frequently failed to connect reads into longer contigs. This suggests that the overlap detection and cycle removal steps may be overly stringent, breaking potential connections between reads. However, we observed promising exceptions: when $N \geq 3162$ with appropriate $k$ values, and particularly when $k = 0$ for moderate coverage values ($C \in \{0.92, 2, 5\}$), N50 values significantly exceeded read length. This demonstrates that our basic algorithm has substantial potential when parameter selection is optimized.

#### The $k$-mer Optimization Trade-off

Our $k$-mer filtering approach was implemented to improve computational efficiency, but the results suggest it introduces significant trade-offs. While higher $k$ values (10, 15) reduced computational demand, they also severely limited the overlap graph's connectivity, often preventing the formation of longer contigs. Conversely, $k = 0$ (no filtering) produced dramatically better results for most metrics at lower coverage values, with N50 values up to ten times the read length and contig counts reduced by an order of magnitude. However, this came at the cost of prohibitive computational requirements for higher coverage scenarios ($C = 10, 30$), highlighting a trade-off between efficiency and assembly quality.

#### The Coverage Paradox

Contrary to expectations, increasing coverage (through higher $N$ or $l$ values) did not consistently improve assembly quality. Beyond a certain threshold, both mismatch rates and contig counts increased with coverage, suggesting that our approach is vulnerable to error accumulation and graph complexity at higher coverage levels. This finding contrasts with the traditional assumption that higher coverage invariably leads to better assemblies and highlights the need for more sophisticated error correction and graph simplification methods.

#### Optimal Parameter Selection

Our results suggest that optimal assembly performance is achieved under specific parameter combinations rather than at extreme values. Moderate error probabilities ($p = 0.001$ or $p = 0.01$) consistently yielded better outcomes than high error probabilities ($p = 0.1$). Similarly, intermediate $k$ values (e.g., $k = 5$) struck an effective balance between computational efficiency and overlap graph connectivity, particularly at higher coverage levels. These findings underscore the importance of careful parameter optimization in genome assembly.

**Limitations**

Several limitations affected our study and the performance of our assembly approach:

1. Graph Simplification Strategy: Our current cycle removal strategy, which is based on eliminating the weakest edge in any detected cycle, may be suboptimal—especially in complex graphs. More sophisticated approaches that consider the context of each edge and the overall topology of the graph might better preserve important connections while still effectively eliminating cycles. Alternatively, adopting the "Linear Transitive Reduction Algorithm" of Myers (2005) could offer a more efficient solution.

2. Computational Scalability: The original alignment-based approach proved computationally intensive for larger datasets, necessitating the $k$-mer optimization. However, this optimization significantly impacted assembly quality. Future implementations should explore more efficient alignment algorithms or indexing structures.

3. Error Model Simplicity: Our error model considered only substitutions, while real sequencing errors include insertions and deletions. Additionally, our scoring scheme used fixed values regardless of $p$, potentially limiting adaptability to different error rates.

4. Linear Genome Assumption: Treating the circular $Phi\ X\ 174$ genome as linear may have affected assembly of reads spanning the origin, potentially creating artificial breaks in the assembly.

**Future Directions**

Based on our findings, several avenues for improvement emerge:

1. Adaptive Parameter Selection: Developing methods to dynamically adjust parameters (particularly $k$) based on local graph properties could maintain computational efficiency while preserving important overlaps.

2. Graph Traversal Optimization: Alternative path finding algorithms that consider global graph structure might produce better contig assemblies than our current greedy approach.

3. Parallelization and Optimization: Exploring more efficient implementations, including GPU acceleration for alignment operations, could make unfiltered ($k$ = 0) approaches feasible for higher coverage scenarios.

**Conclusion**

Our investigation demonstrates that overlap-graph-based assembly remains a viable approach for genome reconstruction from short reads, particularly when parameter selection is optimized for specific sequencing conditions. While de Bruijn graph methods have become more common for very short reads, our results suggest that overlap graphs can perform well with modern read lengths when implemented with appropriate optimizations.

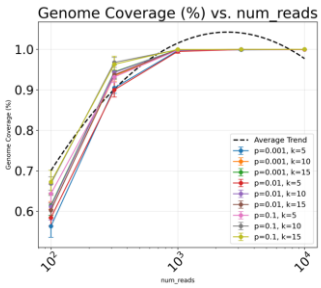
The surprising effectiveness of unfiltered overlap detection (k = 0) at moderate coverage levels highlights the potential of classic overlap-layout-consensus approaches when computational constraints can be overcome. Future work should focus on maintaining this performance while improving algorithmic efficiency, particularly for datasets with higher coverage and error rates.

This study underscores the complex relationship between sequencing parameters and assembly quality, challenging simplistic assumptions about coverage and highlighting the continued importance of methodological refinement in computational genomics.

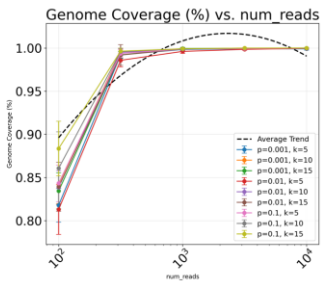
**Appendices**

Appendix_1 – Genome's Coverage values between different fixed $l = 50, 100, 150$:

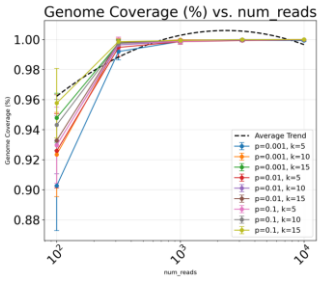
$l = 50$



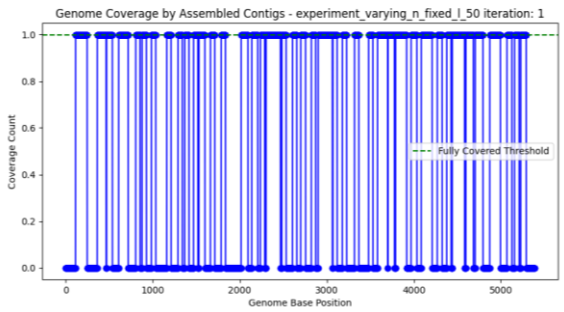

$l = 100$

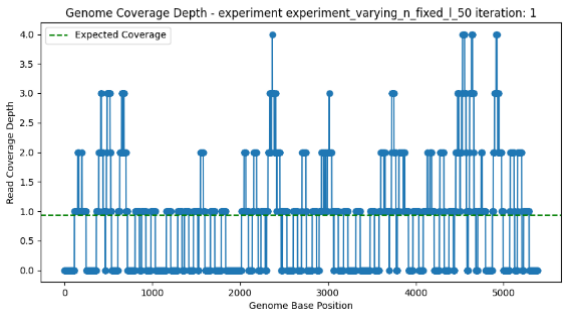



$l = 150$

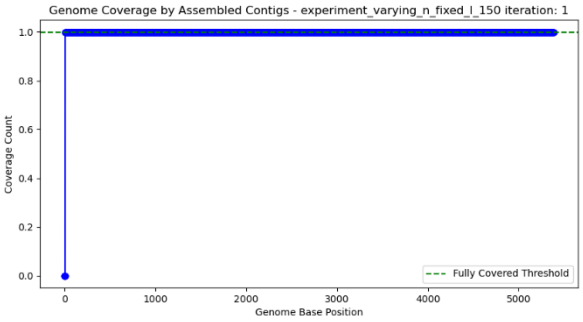


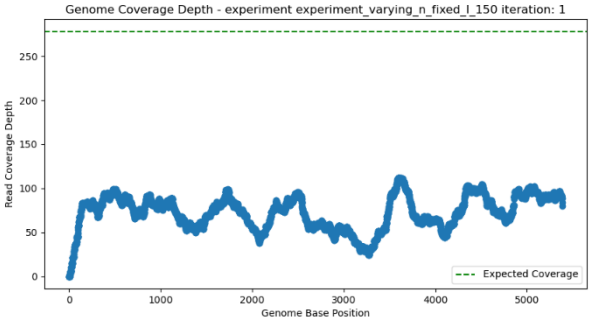

Appendix_1_A – Coverage rate and coverage depth of the genome for $N = 100, l = 50$ with $p = 0.001, k = 5$, not all the bases are covered (left), and some bases are covered by more than 1 contig, meaning the genome would not be covered uniformly or that our contigs aligned mistakenly to other regions than the one they should aligned with.



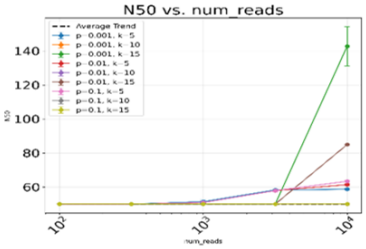

Appendix_1_B – Coverage rate and coverage depth of the genome for $N = 10000, l = 150$ with $p = 0.001, k = 5$, most of the bases are covered, and the genome depth is much larger.



Appendix_2 – N50 values and the trend line for fixed $l = 50, 100, 150$, for all average $N50 = l$:

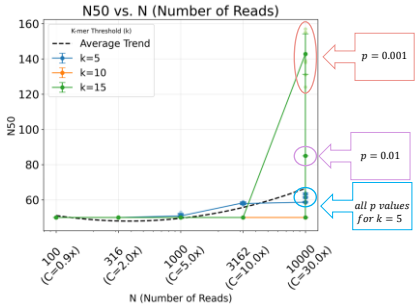
$l = 50$




$l = 100$

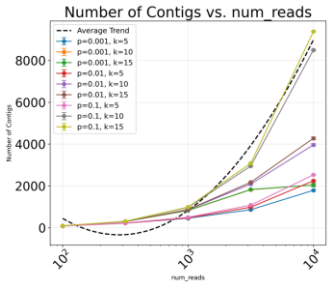

$l = 150$


Appendix_3 – Unexpected result: N50 values for $l = 50$, $k = 15$ obtained the largest N50 value:

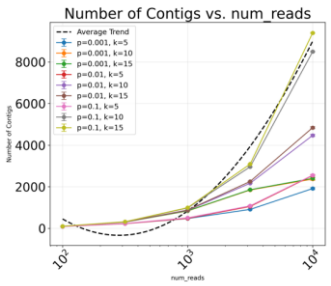



Appendix_4 – Unexpected result: The number of contigs increased as N increased for any $l \in \{50, 100, 150\}$:

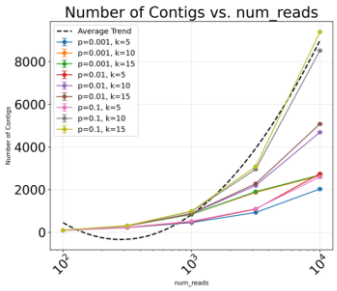
$l = 50$



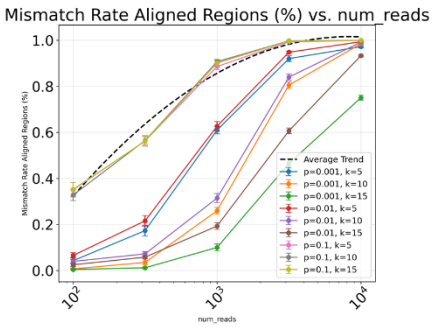

$l = 100$



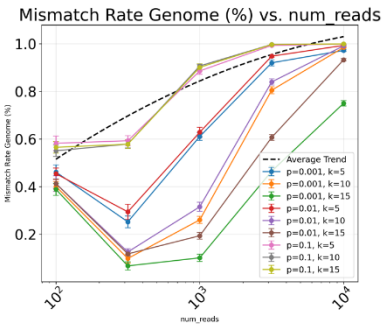

$l = 150$

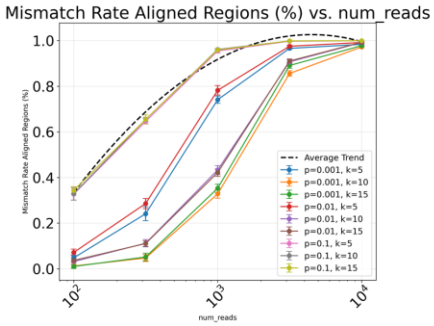



Appendix_5 – Unexpected result: Mismatch rates increased as N increased for any $l$:

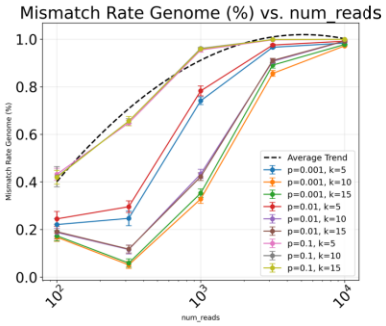


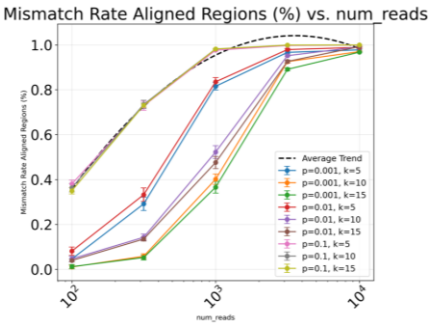

$l = 50$



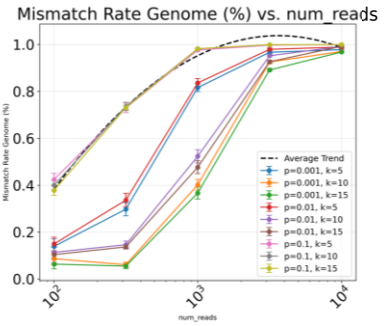



$l = 100$



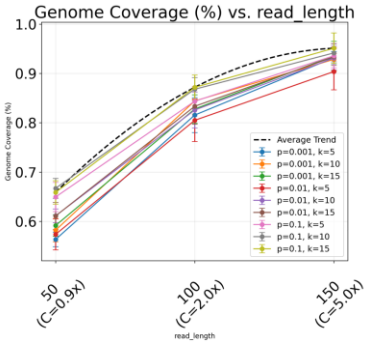




$l = 150$

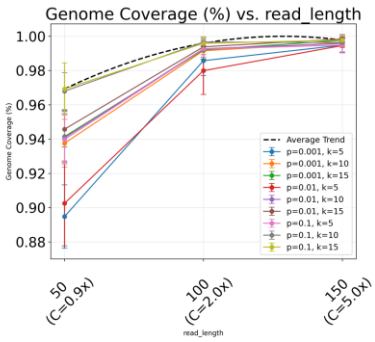



Appendix_6 – genome coverage increased as $l$ increased for any $N$:

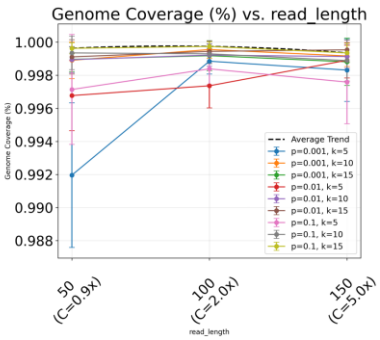
$N = 100$



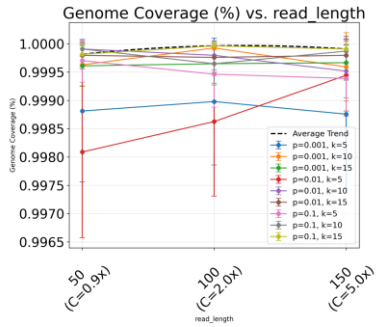

$N = 316$



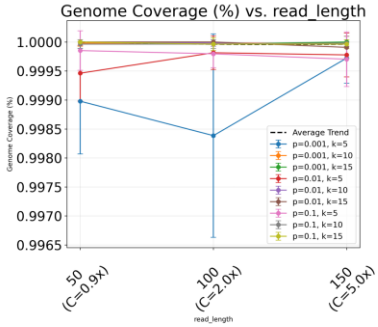

$N = 1000$




$N = 3162$

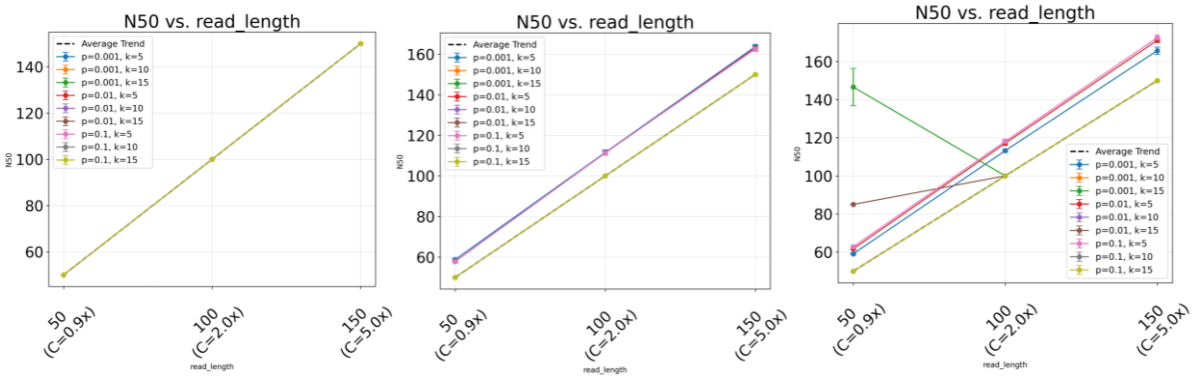



$N = 10000$



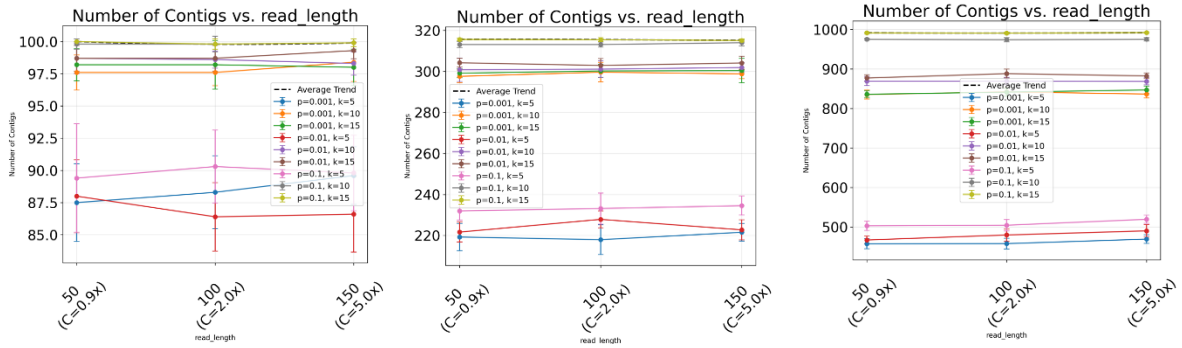
Appendix_7 – N50 for fixed N, only for $N \geq 3162$ we start to see our algorithm assemble contigs as expected, the yellow line is $N50 = l$ that presents the disability of our algorithm to assemble contigs that are longer than the reads:

$N = 100, 316, 1000$ $N = 3162$ $N = 10000$

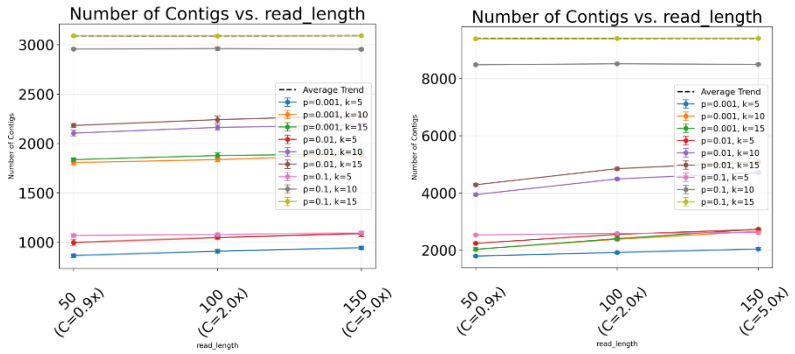



Appendix_8 – The number of contigs is not much affected by the varying of the read length as described here, for each setting (different color in a different plot), the number of contigs stays mostly constant.

$N = 100$ $N = 316$ $N = 1000$

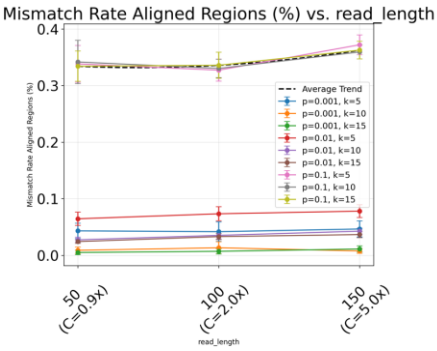


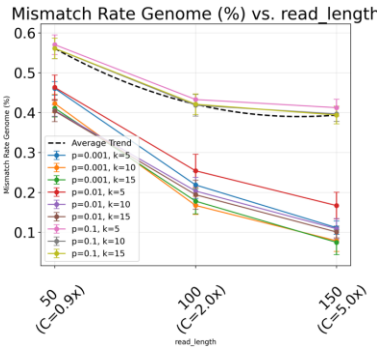

$N = 3162$ $N = 10000$



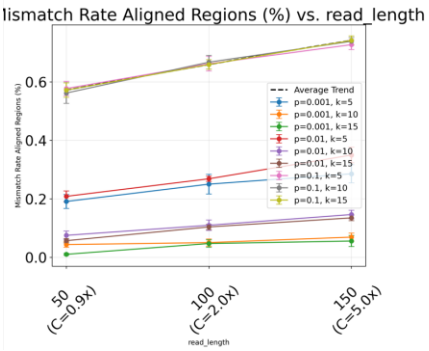
Appendix_9 – The mismatch rates (aligned regions and whole genome) for different $l$ and fixed $N$, some of them, mostly from small $N$ values provide good mismatch rates (means good assembly).

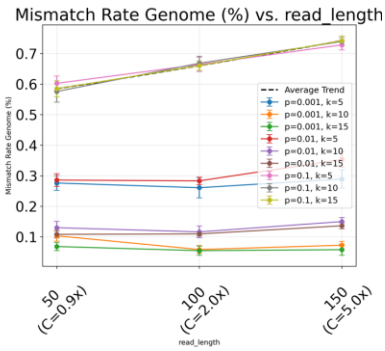
$N = 100$



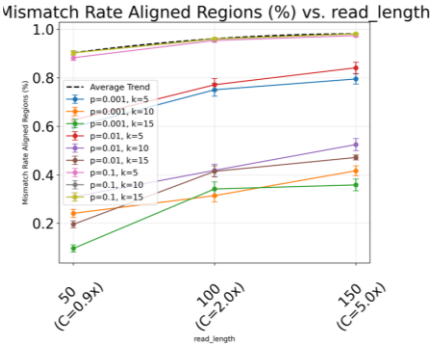



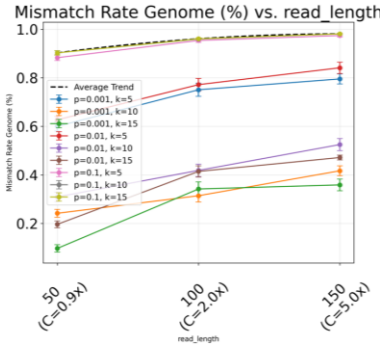

$N = 316$



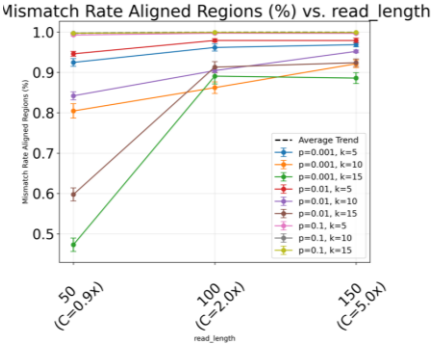



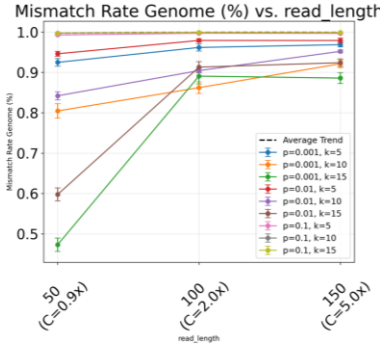

$N = 1000$



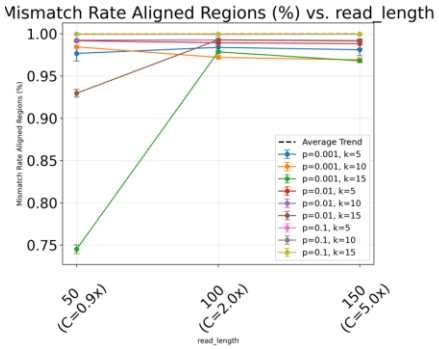



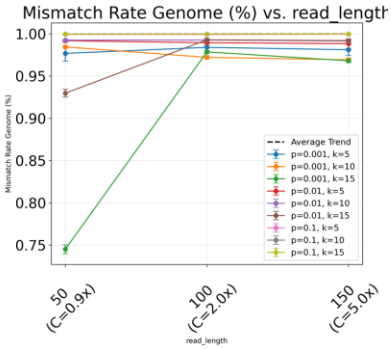

$N = 3162$



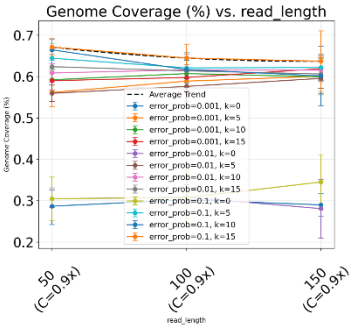
$N = 10000$



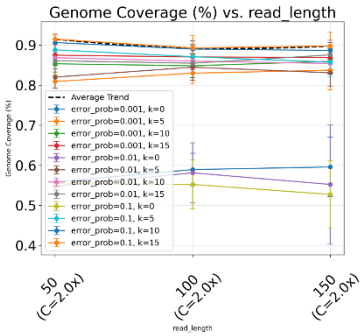




Appendix_10 – the genome coverage full fixed $C$, expected to be almost 1 for $C \geq 1$:

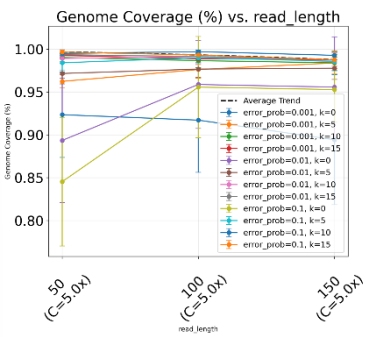
$C = 0.92$



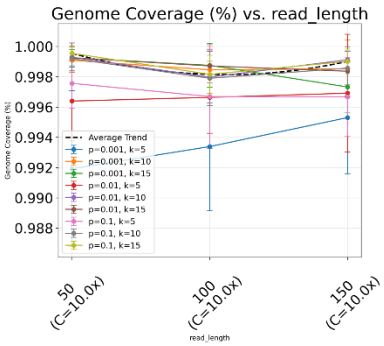

$C = 2$



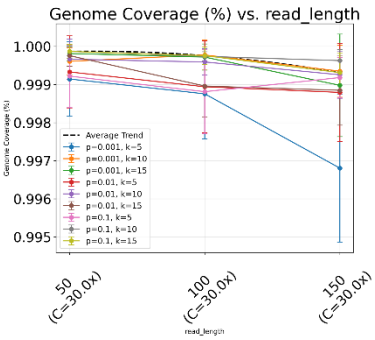

$C = 5$




$C = 10$

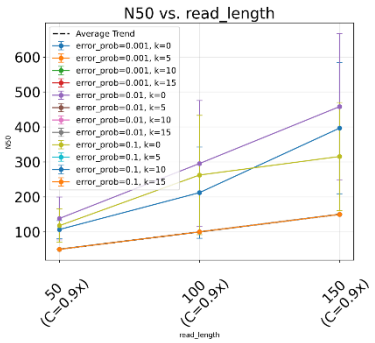



$C = 30$

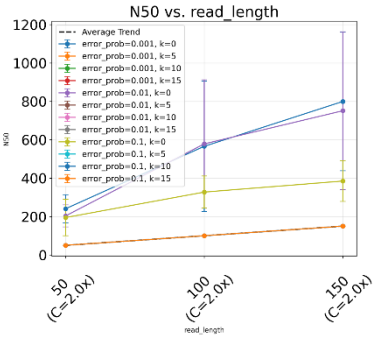


Appendix_11 – N50 as a function of increasing read length and reducing number of reads, we can see that $k = 0$ significantly improved the performance but we didn't have its performance over $C = 10$ and $C = 30$ because it was to heavy to compute. Also $K = 5$ for $C = 30$ achieved $N50 > l$:

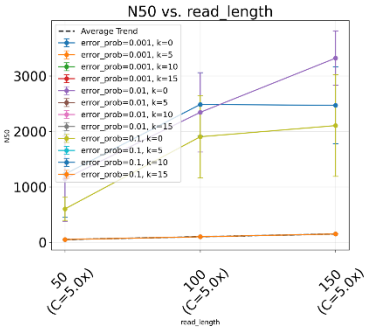
$C = 0.92$



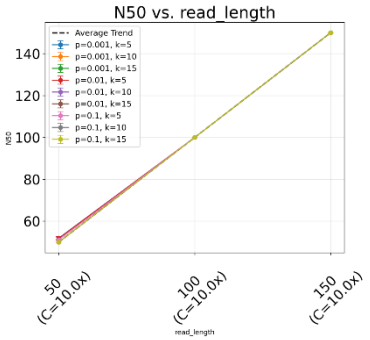

$C = 2$



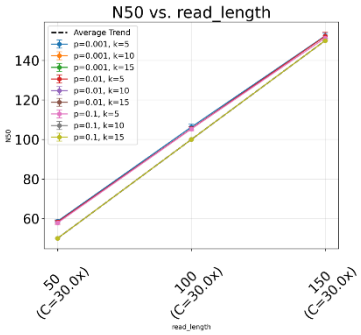

$C = 5$




$C = 10$

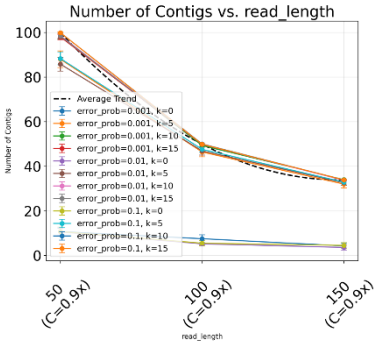



$C = 30$

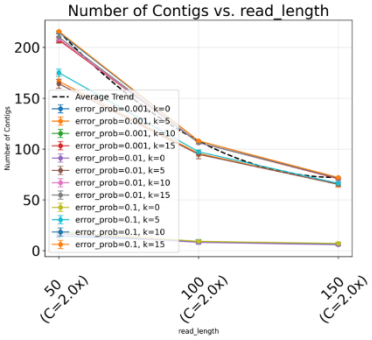


Appendix_12 – the number of contigs as a function of increasing read length and decreasing number of reads (constant genome coverage), we can observe that for $k = 0$ the number of contigs was at least 10 times lower than for any other $k$ across all $l$ values (only for $C \in \{0.92, 2, 5\}$) and that generally lower $k$ value outperformed higher values:

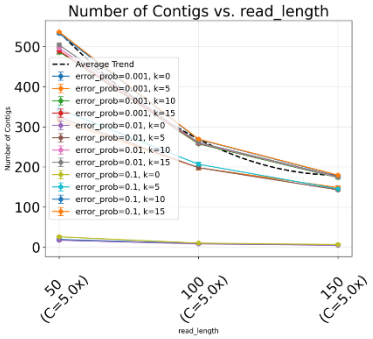
$C = 0.92$



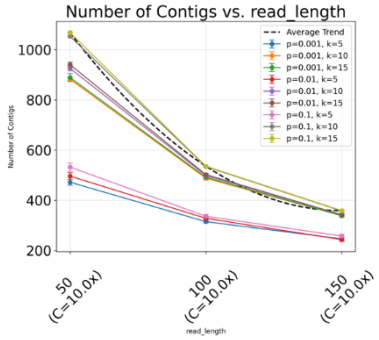

$C = 2$



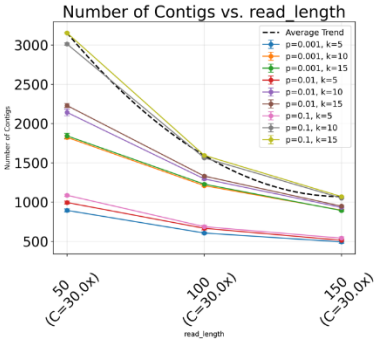

$C = 5$




$C = 10$

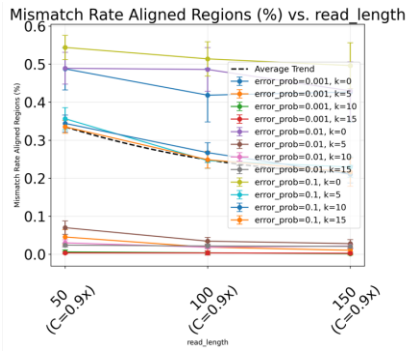


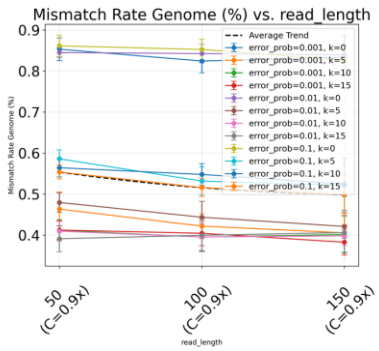

$C = 30$



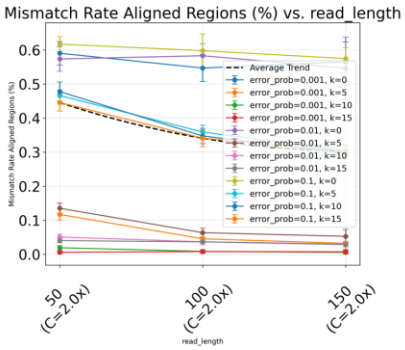

Appendix_13 – mismatch rate as a function of read length, it can be seen that as $l$ increased the mismatch rates decreased, and as $N$ increased (by the increasing of C for the same $l$) the mismatch rates increased. As lower the $k$ is better the mismatch rates.

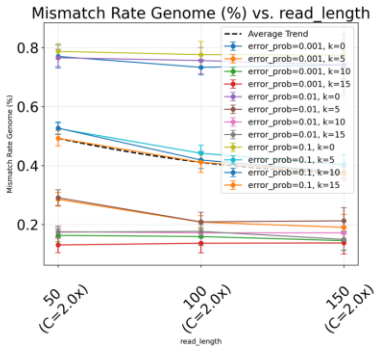
$C = 0.92$



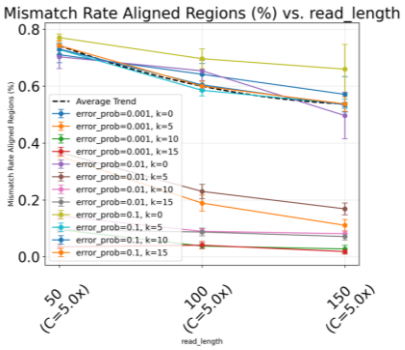



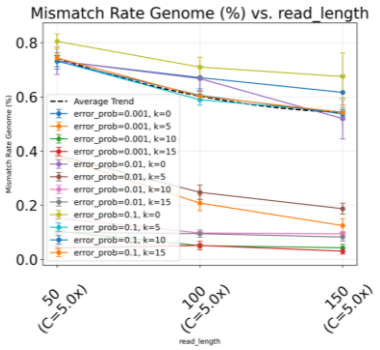
$C = 2$


Mismatch Rate Aligned Regions (%) vs. read_length
Mismatch Rate Aligned Regions (%)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Average Trend
error_prob=0.001, k=0
error_prob=0.001, k=5
error_prob=0.001, k=10
error_prob=0.001, k=15
error_prob=0.01, k=0
error_prob=0.01, k=5
error_prob=0.01, k=10
error_prob=0.01, k=15
error_prob=0.1, k=0
error_prob=0.1, k=5
error_prob=0.1, k=10
error_prob=0.1, k=15
50 (C=2.0x)
100 (C=2.0x)
150 (C=2.0x)
read_length

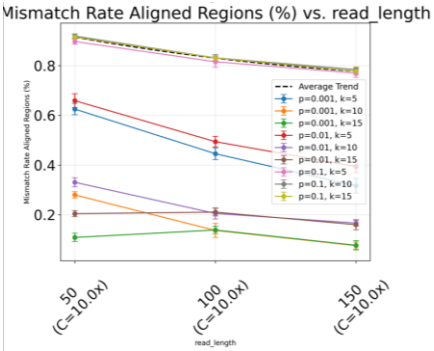

Mismatch Rate Genome (%) vs. read_length
Mismatch Rate Genome (%)
0.2
0.4
0.6
0.8
Average Trend
error_prob=0.001, k=0
error_prob=0.001, k=5
error_prob=0.001, k=10
error_prob=0.001, k=15
error_prob=0.01, k=0
error_prob=0.01, k=5
error_prob=0.01, k=10
error_prob=0.01, k=15
error_prob=0.1, k=0
error_prob=0.1, k=5
error_prob=0.1, k=10
error_prob=0.1, k=15
50 (C=2.0x)
100 (C=2.0x)
150 (C=2.0x)
read_length

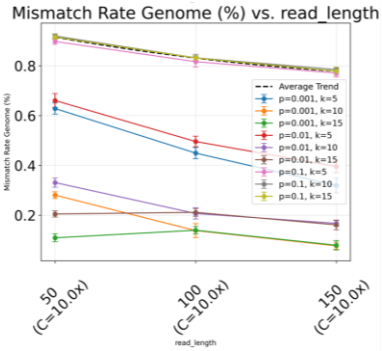
$C = 5$


Mismatch Rate Aligned Regions (%) vs. read_length
Mismatch Rate Aligned Regions (%)
0.0
0.2
0.4
0.6
0.8
Average Trend
error_prob=0.001, k=0
error_prob=0.001, k=5
error_prob=0.001, k=10
error_prob=0.001, k=15
error_prob=0.01, k=0
error_prob=0.01, k=5
error_prob=0.01, k=10
error_prob=0.01, k=15
error_prob=0.1, k=0
error_prob=0.1, k=5
error_prob=0.1, k=10
error_prob=0.1, k=15
50 (C=5.0x)
100 (C=5.0x)
150 (C=5.0x)
read_length

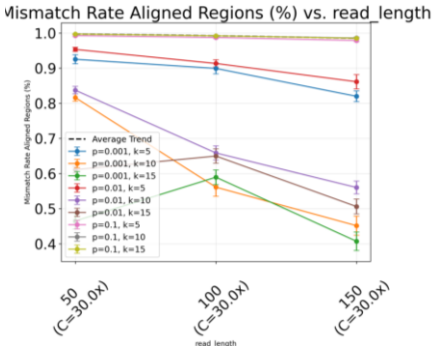

Mismatch Rate Genome (%) vs. read_length
Mismatch Rate Genome (%)
0.0
0.2
0.4
0.6
0.8
Average Trend
error_prob=0.001, k=0
error_prob=0.001, k=5
error_prob=0.001, k=10
error_prob=0.001, k=15
error_prob=0.01, k=0
error_prob=0.01, k=5
error_prob=0.01, k=10
error_prob=0.01, k=15
error_prob=0.1, k=0
error_prob=0.1, k=5
error_prob=0.1, k=10
error_prob=0.1, k=15
50 (C=5.0x)
100 (C=5.0x)
150 (C=5.0x)
read_length

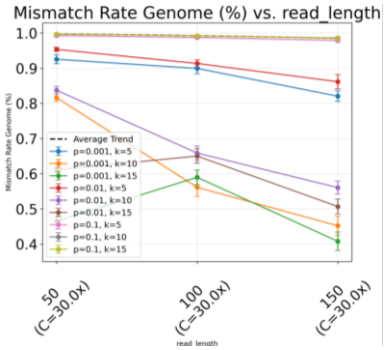
$C = 10$


Mismatch Rate Aligned Regions (%) vs. read_length
Mismatch Rate Aligned Regions (%)
0.2
0.4
0.6
0.8
Average Trend
p=0.001, k=5
p=0.001, k=10
p=0.001, k=15
p=0.01, k=5
p=0.01, k=10
p=0.01, k=15
p=0.1, k=5
p=0.1, k=10
p=0.1, k=15
50 (C=10.0x)
100 (C=10.0x)
150 (C=10.0x)
read_length


Mismatch Rate Genome (%) vs. read_length
Mismatch Rate Genome (%)
0.2
0.4
0.6
0.8
Average Trend
p=0.001, k=5
p=0.001, k=10
p=0.001, k=15
p=0.01, k=5
p=0.01, k=10
p=0.01, k=15
p=0.1, k=5
p=0.1, k=10
p=0.1, k=15
50 (C=10.0x)
100 (C=10.0x)
150 (C=10.0x)
read_length

$C = 30$


Mismatch Rate Aligned Regions (%) vs. read_length
Mismatch Rate Aligned Regions (%)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Average Trend
p=0.001, k=5
p=0.001, k=10
p=0.001, k=15
p=0.01, k=5
p=0.01, k=10
p=0.01, k=15
p=0.1, k=5
p=0.1, k=10
p=0.1, k=15
50 (C=30.0x)
100 (C=30.0x)
150 (C=30.0x)
read_length


Mismatch Rate Genome (%) vs. read_length
Mismatch Rate Genome (%)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Average Trend
p=0.001, k=5
p=0.001, k=10
p=0.001, k=15
p=0.01, k=5
p=0.01, k=10
p=0.01, k=15
p=0.1, k=5
p=0.1, k=10
p=0.1, k=15
50 (C=30.0x)
100 (C=30.0x)
150 (C=30.0x)
read_length

**References**


[1] Shendure, Jay, and Hanlee Ji. "Next-generation DNA sequencing." *Nature biotechnology* 26.10 (2008): 1135-1145.

[2] Myers, Eugene W. "The fragment assembly string graph." Bioinformatics 21.suppl_2 (2005): ii79-ii85.

[3] Lander, Eric S., and Michael S. Waterman. "Genomic mapping by fingerprinting random clones: a mathematical analysis." Genomics 2.3 (1988): 231-239.

[4] Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid
sequence of two proteins. Journal of molecular biology, 48(3), 443-453.

[5] Simpson, Jared T., and Richard Durbin. "Efficient de novo assembly of large genomes using compressed data structures." *Genome research* 22.3 (2012): 549-556.

[6] Shendure, J. *et al*. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science* 309, 1728-1732 (2005).

[7] Xia, Xuhua. "Contig assembly." Bioinformatics and the Cell: Modern Computational Approaches in Genomics, Proteomics and Transcriptomics (2007): 49-61.

[8] Shannon, Claude E. "A mathematical theory of communication." The Bell system technical journal 27.3 (1948): 379-423.

[9] Jenike, Katharine M., et al. "k-mer approaches for biodiversity genomics." Genome Research 35.2 (2025): 219-230.

[10] Smith, Temple F., and Michael S. Waterman. "Identification of common molecular subsequences." Journal of molecular biology 147.1 (1981): 195-197.