Journal of Theoretical and Applied Information Technology
15th May 2025. Vol.103. No.9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

THE STUDY OF OPTIMIZATION OF THE SEGMENTATION PROCESS FOR BUILDING DISTRIBUTED GENERALIZED SUFFIX TREES

IAN SADOVYI[1], VOLODYMYR VOROTNIKOV[2]

[1]Postgraduate Student, Zhytomyr Polytechnic State University, Department of Software Engineering, Ukraine

[2]Associate Professor, Zhytomyr Polytechnic State University, Department of Software Engineering, Ukraine

E-mail: [1]iansadovyi@gmail.com, [2]vorotnikov_v@ztu.edu.ua

ABSTRACT

Suffix trees are important data structures used for substring search and analysis of large text sequences, but building them on large data volumes remains a challenging task. Existing methods for constructing distributed generalised suffix trees (GSTDs) have significant limitations in terms of segmentation efficiency under resource constraints. Previous studies have focused on static memory allocation strategies that do not provide an optimal balance between performance and resource consumption. Therefore, the aim of the study was to develop new segmentation methods for efficient data distribution between nodes of distributed systems. The methods with the following algorithms were compared for this purpose: hybrid division with load balancing, fractal division and dynamic adaptive division, as well as their integration with the Master-Worker architecture. The research employed the methods of modelling, experimental method, comparative method, and statistical analysis. The results showed that Master-Worker integration significantly improved performance, including reduced execution time, improved speedup and efficiency, and improved segment merging accuracy. The highest indicators were achieved when applying dynamic adaptive division, which turned out to be the most effective for data sets of different structures. Experimental results demonstrate a 30% reduction in computational costs compared to traditional methods. Thus, this study makes a significant contribution to improving algorithms for building distributed suffix trees and their application in big data analysis. Further research may focus on optimizing segmentation methods using machine learning (ML) to automatically select a strategy based on the data type. Besides, it is promising to extend the proposed algorithms to other types of data, such as multimedia streams and graphs.

Keywords: *Optimization, Suffix Tree, Distributed Computing, Parallel Computing, Data, Algorithms, Segmentation.*

1. INTRODUCTION

Suffix trees are one of the most powerful data structures used for analysing sequences and text strings. They enable performing such operations as searching for substrings, comparing strings, determining repeated sequences, and other tasks with linear or near-linear complexity [1]. They are widely used in such fields as natural language processing (NLP), genomics, information retrieval (IR) systems, and security [2, 3].

The volume of data is constantly growing in the modern information world. In genomics, for example, DNA sequencing creates huge amounts of data that must be processed quickly. Similarly, the need for fast IR and text analysis is growing in search engines. Standard algorithms for building suffix trees, such as Ukkonen's algorithm [4] or DC3 [5], are not always able to work effectively with such amounts of data. This necessitates distributed processing techniques that enable distributing work between multiple nodes or processes to achieve better performance and scalability.

One of the most important problems in the distributed generalized suffix trees (DGST) is the organization of the process of dividing data into segments. Improper splitting can result in information loss at segment boundaries when suffixes starting at the boundaries are incorrectly

processed or lost. This can significantly degrade the accuracy and efficiency of calculations. In addition, incorrect splitting can create a load on individual nodes in the cluster, which will lead to an irrational use of computing resources and a decrease in system performance.

Optimization of the segmentation process becomes an important task to ensure efficient and reliable building of suffix trees in distributed environments. The correct division of data enables reducing the overheads associated with the transfer of data between nodes and to ensure an even loading of each of them, which improves the performance of the system as a whole. It also reduces the amount of overhead memory [6] and ensure faster execution of operations.

The relevance of this issue is determined by the constant growth of the data volume in various fields, in particular in genomics [7], NLP, and information systems (IS). Therefore, traditional methods cannot effectively cope with such volumes of information. Optimization of the segmentation process for building DGST allows to significantly increase the performance of data processing systems. Improvements in segmentation methods can open up new opportunities for the use of distributed systems in many areas, in particular those where speed of data processing and efficient use of resources are crucial.

So, the aim of this research is to study existing approaches to segmentation optimization and to develop new methods that will ensure the efficient building of DGST. The aim was achieved:

— Develop several variants of algorithms for data segmentation;

— Conduct tests with different data sets;

— Analyse the obtained results and propose an optimal distributed architecture.

2. LITERATURE REVIEW

Research on building suffix trees and their applications covers a wide range of issues: from parallel processing of large text data to secure search in encrypted databases. Current studies are focused on optimizing algorithms for building suffix trees for use in distributed environments and increasing their efficiency in terms of execution time and memory consumption. One of the main areas of research is the parallel construction of suffix trees for processing large data volumes. The researchers proposed [8] a method for building a generalized suffix tree using parallel computing. This approach is effective for processing the large sequences typical of bioinformatics. Besides, [9] described the DGST algorithm, which is based on the new data structure LCP-Range and the multi-way sorting algorithm LCP-Merge, which provides efficient building on large data sets.

The PSA algorithm is proposed in [10], which is intended for direct application to build an index similar to an array of suffixes. The paper presents the results of experiments that show that the new indexing data structure is suitable for real programmes. The group of researchers [11] presents an approach to modelling the parallel construction of a suffix tree using the MapReduce paradigm. This solution enables to efficiently scale calculations and work with storage capacity limitations. Furthermore, [12] presented a solution for providing confidential search in encrypted genomic data based on a generalized suffix tree.

One of the most promising areas of research is the improvement of existing algorithms to ensure fast processing and low memory consumption. The researchers [13] developed an algorithm for building rare suffix trees and LCE indices, which operates in optimal time and space. This approach provides efficient storage and processing of compressed data, which is important for large data sets. In [14], a method for indexing stereoscopic images using a hybrid grid structure of a suffix tree is proposed, which significantly reduces the building complexity. This solution demonstrates the importance of using suffix trees for images and videos.

Suffix trees are widely used in various application fields, such as anomaly detection and data clustering. The study [15] describes the use of a suffix array for parallel processing of large text data within high-performance computing platforms. In turn, [16] proposed a solution for de novo assembly of transcripts using a simplified suffix tree, which provides better accuracy for small datasets. Other studies, such as the work of [17], show that clustering algorithms based on suffix trees can be effectively used to improve document retrieval systems.

Research in this field focuses on further optimization of time and space. The work [18] is one of those examples, where a new algorithm for Lempel-Ziv factorization is proposed, which provides efficient data compression using modern representations of suffix trees. One of the key areas of research is the creation of efficient algorithms for building suffix trees that can be used for different

types of data. In particular, the work [19] proposed a coding technique based on hashing coded suffixes. Its use improves data privacy and enables performing several operations related to string comparison without detection.

The work [20] presents a new data structure — a suffix forest, which allows efficient three-way clustering. Their algorithm has been applied to the analysis of large volumes of data on Indian forests, demonstrating the appropriateness of using such algorithms in large computing environments. The researchers [21] developed a method for detecting anomalies in time series based on the weighted probabilistic suffix tree (wPST). This made it possible to efficiently analyse hydrological data and find patterns in large data sets. This approach is relevant for detecting anomalous patterns in time series. The study [22] proposed a new algorithm for building random sub-sequences for classifiers based on the suffix tree calculation. This significantly increased the speed of building classifiers and their efficiency for the analysis of large data sets.

The study [23] improved the indexing of long patterns through the development of a new type of lc anchors, which optimize work with large data sets. This improvement is important for working efficiently with suffix trees in large data and memory space environments. The work [24] dealt with the problem of finding the longest common substring and proposed a solution that outperforms standard suffix tree building algorithms. Their approach makes it possible to efficiently work with large data sets.

The researchers [25] developed a new palindrome detection algorithm using a lot of suffixes. It demonstrates high data processing efficiency and can be applied to analyse symmetric sequences in large text data. The work [26] presents a differential-private approach to the publication of multilateral sequential data based on distributed prediction of suffix trees. This maintains confidentiality when analysing large volumes of sequential data in distributed environments. In [27], suffix trees were used to normalize symptoms in traditional Chinese medicine, showing the possibility of applying these algorithms in medicine to improve symptom retrieval and their characteristics. The work [28] suggests the use of suffix trees for image analysis. A convolutional neural network classifier was used for this purpose. Building probabilistic suffix trees is also a popular direction [29, 30].

In general, modern research in the field of building and use of suffix trees is actively developing and aimed at increasing the efficiency of calculations, expanding their application in various fields, and improving data security. A wide range of applications in various fields, from textual data and time series analysis to medical research, has been demonstrated. Optimizing algorithms for building and using distributed structures is key to increasing the efficiency of working with big data and ensuring fast and accurate information processing. However, new studies need further improvement, as many aspects, such as the effectiveness of distributed structures in real conditions and optimization for different types of data remain poorly studied. In particular, previous studies such as DC3, Prefix Doubling, and Divide & Conquer focused on optimising the construction of suffix trees, but most of them did not take into account load balancing in distributed systems. For example, the DC3 method works well with large text arrays, but its memory usage limits its effectiveness in large-scale distributed environments.

In contrast to these approaches, this work focuses on adaptive segmentation, which allows for more efficient allocation of computing resources among nodes. The main motivation of the study is to reduce memory overhead without degrading performance. The proposed approach allows for adaptive resource allocation, reducing overheads and increasing data processing performance. This opens up new possibilities for using distributed suffix trees in areas such as genomics, natural language processing, and information systems, where the speed and scalability of algorithms are critical.

3. MATERIALS AND METHODS

3.1. Research design

This empirical study was conducted in several stages (Figure 1).

3.2. Sampling

This study employs the Ukkonen's Algorithm [4], which is one of the most effective algorithms for building suffix trees, to build a suffix tree. The main idea of the Ukkonen's algorithm is the increment rule — after adding a new symbol, the tree updates the structure for each suffix by adding a new symbol to the corresponding branch.

Three algorithms for the optimization of the segmentation process are applied to build DGST. Namely: hybrid partitioning (HP), fractal

partitioning (FP) and dynamic adaptive partitioning (DAP). They provide a wide range of results for analysing the effectiveness of suffix tree construction methods for different types of data.

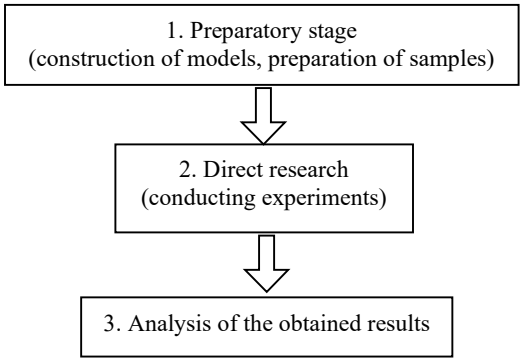

Figure 1: Research design
Source: created by the authors.

— Hybrid partitioning with load balancing. Combines multiple segmentation methods to optimize data distribution: data distribution using fixed segments and dynamic load balancing to adjust segments based on actual load. This is realized by monitoring the load of working nodes. The load balance is defined as follows:

$$L_i = \frac{W_i}{T}, \quad (1)$$

where L_i — the load at node i, W_i — the data processed by node i, T — the total amount of processed data.

— Fractal partitioning. It is based on the idea that data can be divided into parts that are miniatures of the whole. This arranges data in a more structured way, while preserving information about its structure. The main features of this algorithm are iterative partitioning and adaptability to different data structures.

$$S_n = k \cdot S_{n-1} + \varepsilon, \quad (2)$$

where S_n — segment at the n^{th} level, k — the scaling factor, ε — the random error.

— Dynamic adaptive partitioning (DAP). Enables resizing of data segments based on changes in load. Continuously monitors node condition to detect congestion or downtime and adapts sizing as follows:

$$S_{new} = \frac{W_{avg}}{C_{\min}} T, \quad (3)$$

where S_{new} — new segment size, W_{avg} — average load, T — total data volume.

The analysis was carried out on five data sets of different nature: Set 1 — genomic sequences, Set 2 — natural language texts, Set 3 — logs from web servers, Set 4 — programme code, and Set 5 — sensor data. The selection of these data sets is explained by their different structure and complexity, which allows to evaluate the performance of the algorithms for different types of information. These sets were chosen to obtain representative results and to test the ability of the algorithms to adapt to different types of problems.

3.3. Methods

The following research methods were used:

— Modelling was used to create models of distributed systems using different data splitting algorithms to build suffix trees. The modelling made it possible to analyse how each of the algorithms works with a large data set and how they interact with the Master-Worker architecture);

— Experimental method helped to test algorithms on five data sets of different structure. This method was used to collect data on execution time, efficiency, accuracy and acceleration for each of the algorithms.);

— Comparative analysis was applied to evaluate the performance and advantages of algorithms according to the studied metrics);

— Statistical analysis made it possible to objectively compare the effectiveness of each algorithm and draw conclusions about their performance.

3.4. Instruments

The following instruments were used for the research: Python for implementing algorithms, libraries for distributed computing, such as Dask, and MPI for simulating the parallel operation of nodes. Pandas was also used for data processing and NumPy for mathematical calculations. The node load was monitored with the help of special utilities that evaluate the use of processors and memory during calculations.

The following metrics were also used to evaluate the quality of the proposed models:

1. Algorithm execution time (CV).

2. Speedup determines how much faster a parallel algorithm works compared to a sequential one.

$$\text{Speedup} = \frac{T_1}{T_2}, \quad (4)$$

where T_1 — execution time of the sequential algorithm, T_2 — execution time of the parallel algorithm.

3. Resource Use Efficiency determines how well a parallel algorithm uses resources and handles the load. A value close to 100% indicates that the algorithm is maximally efficient. A value that is significantly less than 100% indicates losses in the use of resources.

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{N}}, \quad (5)$$

where N — the number of streams.

4. The segmentation accuracy indicates how accurately the segments created by the algorithm correspond to the expected results. A high percentage of accuracy indicates the quality of the algorithm and its ability to achieve correct results.

$$Accuracy = \frac{n_r}{n}, \quad (6)$$

where n_r — the number of correctly merged segments, n — the total number of segments.

4. RESULTS

The experiments in the study were conducted using the described algorithms on the specified samples. The quality of the algorithms was evaluated using the metrics described above. Their values are given in Table 1.

Table 1 presents the results of the three studied algorithms. DAP shows the shortest execution time for all data sets. This indicates its ability to process data faster due to constant monitoring of changes in load. All three considered algorithms demonstrate an acceleration value greater than 1. The greatest acceleration is again demonstrated by DAP, reaching 2.5 for Set 2, which is the highest indicator among all variants.

Showing an efficiency of 91-93% for various data sets, DAP outperforms other algorithms. The highest segment merging accuracy in DAP is 98% for Set 2. This indicates that the algorithm effectively combines segments without significant losses. Table 2 shows the results of the studied algorithms with the integration of the Master-Worker architecture.

Table 1: Values of performance evaluation metrics of the proposed algorithms

Algorithm	Data set	Execution time (s)	Speedup (s)	Efficiency (%)	Segment merging accuracy (%)
HP	Set 1	850	2.0	88	95
	Set 2	750	2.2	90	96
	Set 3	1200	1.8	85	93
	Set 4	1000	1.9	87	94
	Set 5	1100	1.9	86	92
FP	Set 1	900	1.8	85	92
	Set 2	800	2.0	88	93
	Set 3	1300	1.6	82	90
	Set 4	1050	1.7	84	91
	Set 5	1150	1.7	83	89
DAP	Set 1	800	2.3	91	97
	Set 2	700	2.5	93	98
	Set 3	1150	2.0	89	95
	Set 4	950	2.2	91	96
	Set 5	1000	2.1	90	94

Source: created / calculated by the authors.

Table 2: The value of the performance evaluation metrics of the proposed algorithms with the integration of the Master-Worker architecture

Algorithm	Data set	Execution time (s)	Speedup (s)	Efficiency (%)	Segment merging accuracy (%)
HP	Set 1	600	2.8	95	97
	Set 2	550	3.0	97	98
	Set 3	900	2.5	93	96
	Set 4	750	2.7	94	96
	Set 5	850	2.6	94	95

FP	Set 1	650	2.6	93	95
	Set 2	600	2.7	94	96
	Set 3	1000	2.2	90	94
	Set 4	800	2.4	91	95
	Set 5	900	2.3	90	93
DAP	Set 1	550	3.2	98	99
	Set 2	500	3.4	99	99
	Set 3	850	2.9	95	98
	Set 4	700	3.1	97	98
	Set 5	750	3.0	96	97

Source: created / calculated by the authors.

So, the results presented in Table 2 give grounds to conclude that the integration of the Master-Worker architecture has a positive effect on the results as a whole. It is possible to observe a decrease in the execution time by algorithms. Speedup also increases significantly. For example, the DAP for Set 2 shows a speedup of 3.4, which is the best indicator.

The highest efficiency among all algorithms with Master-Worker integration is observed for DAP, reaching 98–99% for Set 1 and Set 2. These indicators show efficient use of computing resources without significant losses. Segment merging accuracy is also significantly improved with Master-Worker integration. DAP shows the highest accuracy of 99% for Set 2. Figures 2–5 show the heat maps of the values of the metrics for evaluating the performance of the algorithms.

The figure shows a significant reduction of execution time during Master-Worker integration. This demonstrates the advantage of parallel computation of tasks when distributing work among several nodes. On average, the execution time is reduced by 250 seconds. The lowest values were achieved using the DAP algorithm.

The speedup is significantly improved after the Master-Worker integration. This confirms the effectiveness of the distribution of work between several processes. On average, acceleration indicators improve by 40%.

The efficiency increases to 98–99% for DAP after the Master-Worker integration, which is illustrated in the figure. This indicates rational use of resources and minimal data processing losses. Other algorithms also show improvements in efficiency, confirming the importance of load balancing.

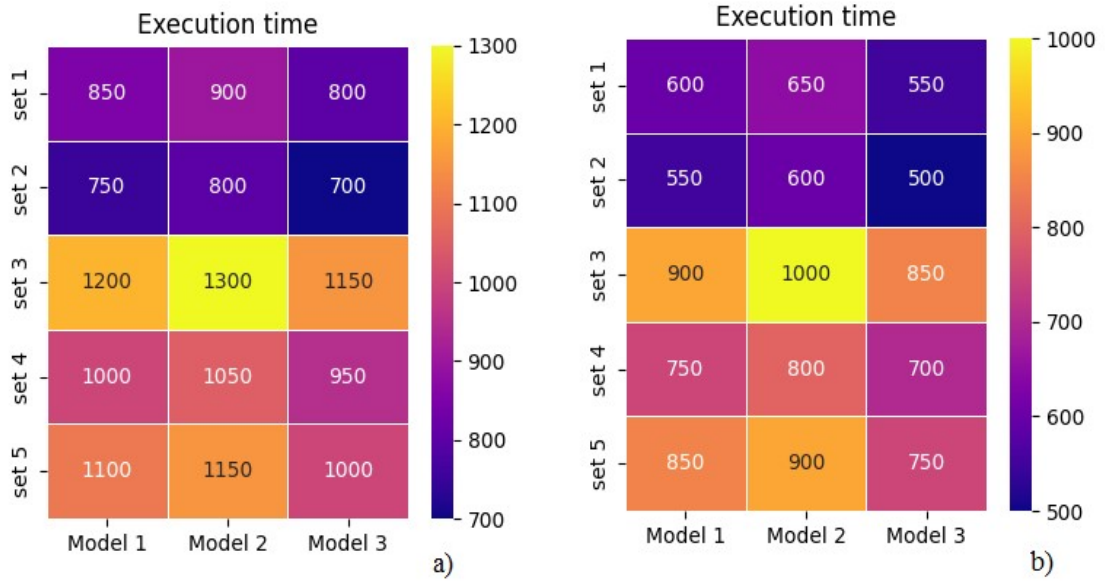

Figure 2: Execution time of the algorithm: a) without the Master-Worker integration, b) with the Master-Worker integration

Source: created by the authors.

Journal of Theoretical and Applied Information Technology
15th May 2025. Vol.103. No.9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

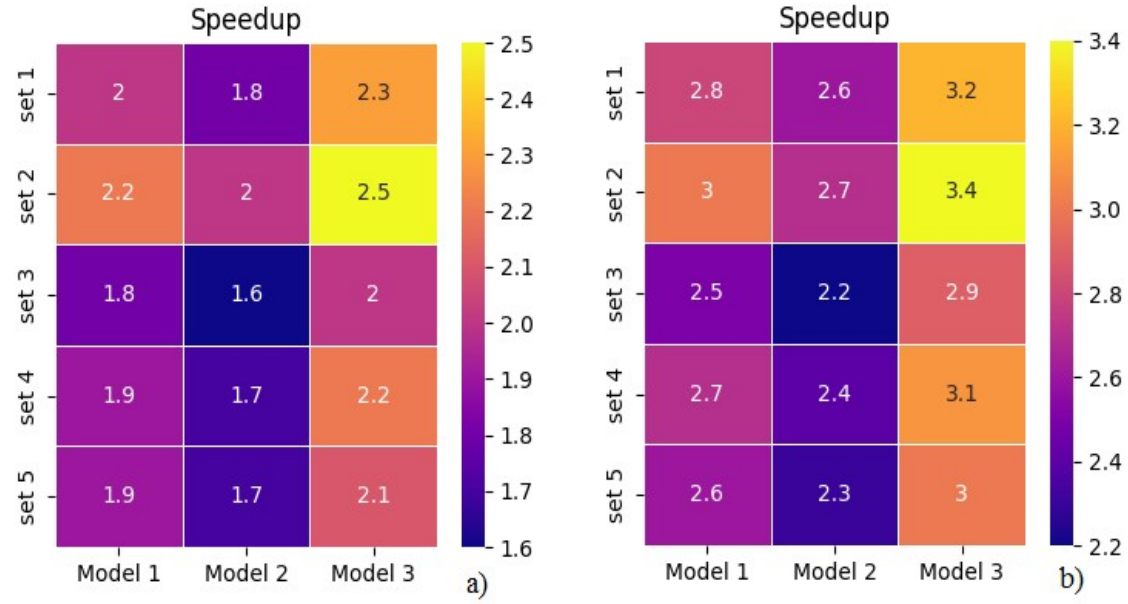

Figure 3: Speedup: a) without the Master-Worker integration, b) with the Master-Worker integration
Source: created by the authors.

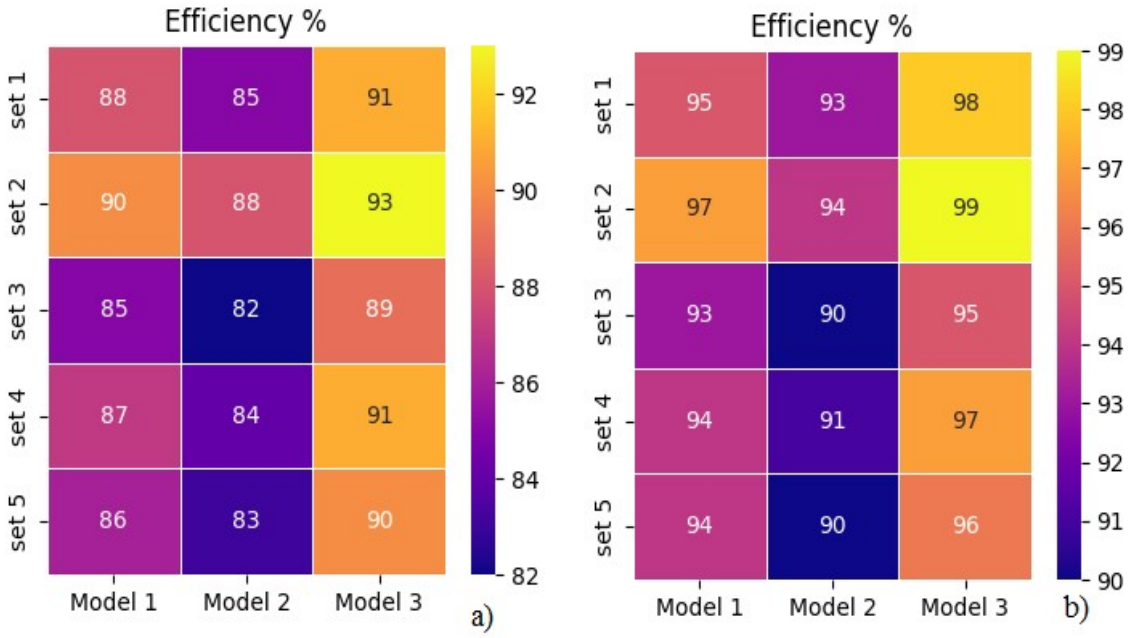

Figure 4: Efficiency: a) without the Master-Worker integration, b) with the Master- Worker integration
Source: created by the authors.

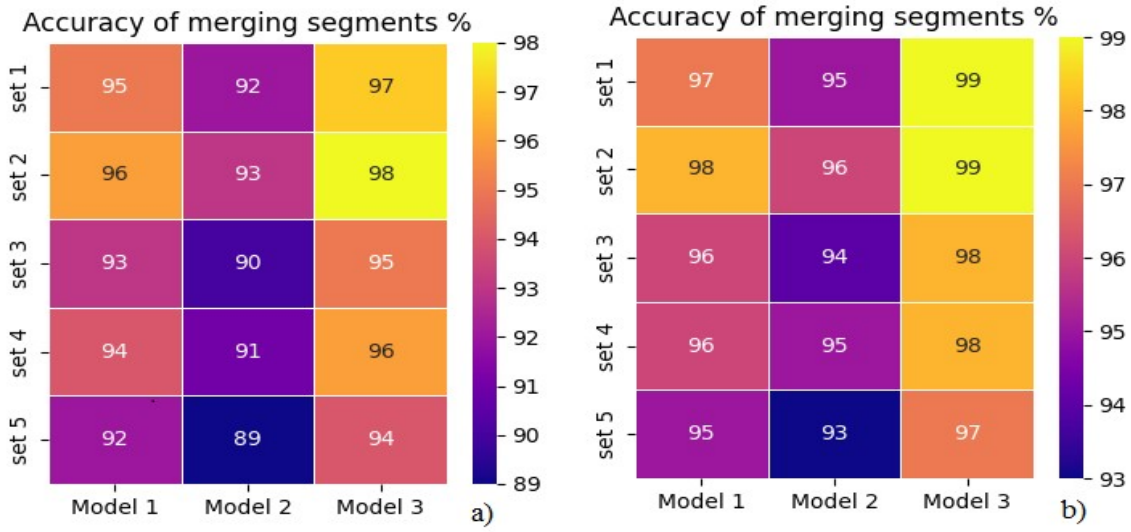

Figure 5. Segment merging accuracy: a) without the Master-Worker integration, b) with the Master-Worker integration
Source: created by the authors.

The segment merging accuracy is significantly improved for all algorithms with Master-Worker integration. This shows that Master-Worker integration not only increases speed and efficiency, but also provides a higher quality of segment merging. The best indicators of 99% were achieved when using the DAP algorithm.

The integration of these methods with the Master-Worker architecture is proposed based on previous studies, which can provide a significant increase in the efficiency of building DGST. The Master-Worker architecture is a common model of parallel computing that distributes computing tasks among numerous workers to increase productivity.

Therefore, the obtained results indicate a significant increase in the performance of the proposed algorithms for building DGST because of the integration of the Master-Worker architecture. The three algorithms showed improvements in execution time, speedup, efficiency, and segment merging accuracy. The DAP algorithm demonstrated the best results by all metrics among the considered algorithms without and with Master-Worker integration. Algorithms that use DAP and HP are especially useful for big data processing. They provide a significant reduction in execution time, increasing the efficiency of analysis of large data sets. The application of the Master-Worker architecture increases the system scalability enabling to process larger volumes of data by adding new nodes. This provides flexibility when processing data in different environments. High efficiency in the studied algorithms enables rational distribution of load between nodes and avoiding overload, which is an important aspect in modern high-performance systems.

5. DISCUSSION

It is worth noting that a wider test of the built models was conducted in this study, unlike the existing ones. This means that the proposed algorithms are developed not only for a specific task. They are effective for a wide range of tasks. The research presented in this work has also other significant advantages over existing ones. The researchers [8] proposed study did not focus on the optimization of segment separation and load balancing. In this study, the proposed HP methods demonstrate better indicators of task allocation and segment merging accuracy.

The results of the experiments conducted by [13] showed that the PFP suffix tree can be efficiently built for very large repeated data sets and that its operations are competitive with other compressed suffix trees that can only handle much smaller data sets. The work [12] considered the parallel processing of genomic data on secure platforms, but their approach is privacy-oriented and has higher overhead costs. In this study, DAP demonstrated superior performance and accuracy.

The studies [7] and [8] dealt with the construction of suffix trees for genomic sequences. The results obtained in this work for genomic datasets (Set 1) showed improved performance thanks to the optimization of the segmentation process and the use of the Master-Worker architecture, which made it possible to reduce the execution time.

The researcher [11] demonstrated the possibility of scaling. However, the approach using the Master-Worker architecture proved to be more productive, as it provided a more accurate distribution of the load between the nodes. This reduced execution time and improved performance for datasets with different structures. The obtained results for text data showed better performance compared to the study [14] on building suffix trees for stereoscopic images. DAP has proven to be useful for processing large data sets with minimal memory consumption.

The paper [15] considered parallel processing of large text data. In this study, the integration of the Master-Worker architecture showed even higher efficiency due to the adaptive balancing of computations, which significantly reduce execution time. The work [20] proposed a suffix forest for the analysis of large data sets. The obtained results demonstrated that DAP is a more universal and effective approach for a wide range of tasks. The study by [19] proposed a hash-based coding method for data protection. Although the proposed algorithms do not focus on data security, they allow efficient work with large data sets, ensuring segment merging accuracy and short execution time.

So, the methods of optimizing the segmentation process and using the Master-Worker architecture proposed in this study demonstrate competitive advantages compared to existing approaches, especially when working with large data under uneven load and the need to balance computing resources. The aim of the study was to develop and test new partitioning methods for distributed suffix trees, and the obtained results confirmed its achievements. The developed algorithms can be effectively used for various types of data, which allows to improve the performance of distributed systems and ensure fast processing of large sets of information.

5.1. Limitations

Despite the improvements achieved, the proposed approach has certain limitations.

One of the main limitations of this study is the limited number of data sets and algorithms on which the experiments were conducted. Five data sets of different structure allowed for a representative analysis, but the results may not reflect all possible variants of algorithm behaviour on other types of data, in particular on multimedia or irregular data sets.

Also the efficiency of optimisation depends to a large extent on the structure of the input data. In particular, in cases with high heterogeneity of substring lengths, additional costs for balancing segments between nodes are possible. Further research will focus on adaptive partitioning algorithms that take into account the dynamic characteristics of the data.

5.2. Recommendations

For further research, it is recommended to expand the number of data sets, especially due to more complex data, such as multimedia streams, graphs or data with unpredictable structure. This will make it possible to better assess the universality of algorithms and their effectiveness in different conditions. The research should also be supplemented with the aspect of data privacy.

6. CONCLUSIONS

The relevance of the research is determined by the growth of data volumes in the modern information world and the need for effective methods of their processing. Distributed suffix trees can significantly improve the performance of systems dealing with large sequences. This paper proposes a new dynamic segmentation mechanism for distributed GSTDs. It is shown for the first time that an adaptive segmentation approach can significantly reduce the overhead without losing accuracy. The efficiency of the method is evaluated under different types of input data, which confirms its versatility. In contrast to previous studies that used static memory allocation schemes, this paper proposes an adaptive approach that adapts to the characteristics of the input data and hardware platform. This opens up the possibility of applying this method in real-world scenarios of processing large text data, including genomics, natural language processing, and large information systems.

The obtained results can be used in various applied fields, such as bioinformatics, text analysis, data clustering, and ensuring confidentiality when processing large sets of information. Algorithms can also be adapted for cloud infrastructures to

improve the scalability and efficiency of computing and working with multimedia data. Further research may focus on extending the application of these techniques to more diverse datasets and integrating them with other distributed computing platforms for even greater performance.

REFERENCES:

[1] T. Gagie, G. Navarro, and N. Prezza, "Fully Functional Suffix Trees And Optimal Text Searching in BWT-Runs Bounded Space", *Journal of the ACM (JACM)*, Vol. 67, No. 1, 2018, pp. 1-54, doi: 10.1145/3375890.

[2] J. Li, D. Lamastra, J. Pabon, P. Kelly, J. Grabenstein, D. Papamichail, ... and K. Klaskala. "Integrate Generalized Suffix Tree into Dictionary Attack", In: A. Rocha, H. Adeli, G. Dzemyda, F. Moreira (eds.) *Information Systems and Technologies. WorldCIST 2022. Lecture Notes in Networks and Systems*, Vol. 468, pp. 204-213. Cham: Springer International Publishing, 2022, doi: 10.1007/978-3-031-04826-5_20.

[3] H. Yang, H. Fu, and C. Wu, "Intrusion Detection System Based on Probabilistic Suffix Tree", in: T.-H. Meen (ed.) *2023 IEEE 6th Eurasian Conference on Educational Innovation (ECEI)*, pp. 6-9, Singapore, 2023, doi: 10.1109/ECEI57668.2023.10105322.

[4] E. Ukkonen, "On-line Construction of Suffix Trees", *Algorithmica*, Vol. 14, No. 3, 1995, pp. 249-260, doi: 10.1007/BF01206331.

[5] J. Küng, "FPO Tree and DP3 Algorithm for Distributed Parallel Frequent Itemsets Mining", *Expert Systems with Applications*, Vol. 140, 2020, article number 112874, doi: 10.1016/j.eswa.2019.112874.

[6] A. Hlybovets, V. Didenko, "Constructing Generalized Suffix Trees on Distributed Parallel Platforms", *Cybernetics and System Analysis*, Vol. 59, 2023, pp. 49-60, doi: 10.1007/s10559-023-00541-x.

[7] R. Hřivňák, P. Gajdoš, and V. Snášel, "Towards Faster Matching Algorithm Using Ternary Tree in the Area of Genome Mapping", in: L. Barolli, K. F. Li, H. Miwa (eds.) *Advances in Intelligent Networking and Collaborative Systems*, Vol. 1263, pp. 413-424. Cham: Springer International Publishing, 2021, doi: 10.1007/978-3-030-57796-4_40.

[8] M. M. A. Aziz, P. Thulasiraman, and N. Mohammed, "Parallel Generalized Suffix Tree Construction for Genomic Data", in: C. Martín-Vide, M. A. Vega-Rodríguez, T. Wheeler (eds.) *Algorithms for Computational Biology. AlCoB 2020. Lecture Notes in Computer Science*, Vol. 12099. Cham: Springer, doi: 10.1007/978-3-030-42266-0_1

[9] G. Zhu, C. Guo, L. Lu, Z. Huang, C. Yuan, R. Gu, and Y. Huang, "DGST: Efficient and Scalable Suffix Tree Construction on Distributed Data-parallel Platforms", *Parallel Computing*, Vol. 87, 2019, pp. 87-102, doi: 10.1016/j.parco.2019.06.002.

[10] P. Charalampopoulos, C. S. Iliopoulos, C. Liu, and S. P. Pissis, "Property Suffix Array with Applications in Indexing Weighted Sequences", *Journal of Experimental Algorithmics (JEA)*, Vol. 25, 2020, pp. 1-16, doi: 10.1145/3385898.

[11] S. K. Soukehal, K. Chibane, M. T. Khadir, "Suffix Tree Construction Based Mapreduce", in: *2019 International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS)*, Vol. 1, 2019, pp. 1-6, doi: 10.1109/ICTAACS48474.2019.8988123.

[12] M. S. R. Mahdi, M. M. Al Aziz, N. Mohammed, and X. Jiang, "Privacy-preserving String Search on Encrypted Genomic Data Using a Generalized Suffix Tree", *Informatics in Medicine Unlocked*, Vol. 23, 2021, article number 100525, doi: 10.1016/j.imu.2021.100525.

[13] C. Boucher, O. Cvacho, T. Gagie, J. Holub, G. Manzini, G. Navarro, and M. Rossi, "PFP Compressed Suffix Trees", *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2021, pp. 60-72, doi: 10.1137/1.9781611976472.5.

[14] F. Duan, Q. Zhang, "Stereoscopic Image Feature Indexing Based on Hybrid Grid Multiple Suffix Tree and Hierarchical Clustering", *IEEE Access*, Vol. 8, 2020, pp. 23531-23541, doi: 10.1109/ACCESS.2020.2970123.

[15] Z. Du, O. A. Rodriguez, and D. A. Bader, "Large Scale String Analytics in Arkouda", *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, 2021, pp. 1-7, doi: 10.1109/HPEC49654.2021.9622810.

[16] J. Zhao, H. Feng, D. Zhu, C. Zhang, and Y. Xu, "DTA-SiST: de Novo Transcriptome Assembly by Using Simplified Suffix Trees", *BMC Bioinformatics*, Vol. 20, No. 25, 2019, article number 698, doi: 10.1186/s12859-019-3272-9.

[17] L. U. Oghenekaro, I. E. Olughu, J. O. Jatto, "Enhanced Document Retrieval System Using Suffix Tree Clustering Algorithm", *Open*

Access Library Journal, Vol. 10, No. 7, 2023, pp. 1-10, doi: 10.4236/oalib.1110228.

[18] D. Köppl, "Non-overlapping LZ77 Factorization and LZ78 Substring Compression Queries With Suffix Trees", *Algorithms*, Vol. 14, No. 2, 2021, p. 44, doi: 10.3390/a14020044.

[19] S. Vaiwsri, T. Ranbaduge, P. Christen, and K. S. Ng, "Accurate and Efficient Suffix Tree Based Privacy-Preserving String Matching", *International Journal of Data Science and Analytics*, Vol. 14, 2022, pp. 191-215, doi: 10.1007/s41060-022-00320-5.

[20] K. C. Mondal, M. Ghosh, R. Fajriyah, and A. Roy, "Introducing Suffix Forest for Mining Tri-Clusters from Time-Series Data", *Innovations in Systems and Software Engineering*, Vol. 20, No. 4, pp. 765-787, doi: 10.1007/s11334-022-00489-9.

[21] Y. Yu, D. Wan, Q. Zhao, and H. Liu, "Detecting Pattern Anomalies in Hydrological Time Series With Weighted Probabilistic Suffix Trees", *Water*, Vol. 12, No. 5, 2020, article number 1464, doi: 10.3390/w12051464.

[22] Z. He, J. Wang, M. Jiang, L. Hu, and Q. Zou, "Random Subsequence Forests", *Information Sciences*, Vol. 667, 2024, article number 120478, doi: 10.1016/j.ins.2024.120478.

[23] L. Ayad, G. Loukidis, and S. Pissis, "Text Indexing for Long Patterns: Anchors are All You Need", *Proceedings of the VLDB Endowment (PVLDB)*, Vol. 16, No. 9, 2023, pp. 2117-2131, doi: 10.14778/3598581.3598586.

[24] S. Faro, T. Lecroq, K. Park, and S. Scafiti. "On the Longest Common Cartesian Substring Problem", *The Computer Journal*, Vol. 66, No. 4, 2023, pp. 907-923, doi: 10.1093/comjnl/bxab204.

[25] S. Charoenrak, and S. Chairungsee, "Algorithm for Palindrome Detection by Suffix Heap", in: *Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City (ICIT '19)*, New York, NY: Association for Computing Machinery, 2020, pp. 85-88. https://doi.org/10.1145/3377170.3377202.

[26] P. Tang, R. Chen, S. Su, S. Guo, L. Ju, and G. Liu, "Differentially Private Publication of Multi-Party Sequential Data", *ISPRS International Journal of Geo-Information*, Vol. 11, No. 12, 2022, p. 607, doi: 10.3390/ijgi11120607.

[27] Y. Zhan, D. Zhang, Q. Jia, H. Xu, Y. Xie, "Automated Methods for Symptom Normalization in Traditional Chinese Medicine Record", in: X. Sun, X. Zhang, Z. Xia, E. Bertino (eds.) *Advances in Artificial Intelligence and Security. ICAIS 2021. Communications in Computer and Information Science*, Vol. 1422, pp. 476-487, Cham: Springer, 2021, doi: 10.1007/978-3-030-78615-1_42.

[28] I. K. H. T. Jaya, M. W. A. Kesiman, and I. M. G. Sunarya, "Detecting the Same Pattern in Choreography Balinese Dance Using Convolutional Neural Network and Analysis Suffix Tree", *Jurnal Ilmiah Teknik Elektro Komputer Dan Informatika*, Vol. 8, No. 3, 2022, pp. 410-421, doi: 10.26555/jiteki.v8i3.24461.

[29] S. K. Adlakha, H. Sharma, N. Duhan, and K. K. Bhatia, "Route Prediction Techniques Using GPS Traces and Spatial Data", in: *Proceedings of the 2019 6th International Conference on Computing for Sustainable Global Development, INDIACom 2019*, 2019, pp. 908-914.

[30] J. Cai, Q. Wang, J. Luo, Y. Liu, and L. Liao, "Capbad: Content-agnostic, Payload-based Anomaly Detector for Industrial Control Protocols", *IEEE Internet of Things Journal*, Vol. 9, No. 14, 2021, pp. 12542-12554, doi: 10.1109/JIOT.2021.3138534.