Applying Myers Difference Algorithm for Version Analysis and Transparency Enhancement in Legislative Document

Ahmad Zaky Robbani - 13524045
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ahmadzakyrobbani@gmail.com, 13524045@std.stei.itb.ac.id

***Abstract*—The process of forming laws and regulations constitutes an important aspect of democratic governance and requires effective public oversight to ensure transparency and accountability. During the legislative process, multiple versions of draft documents may be produced before a regulation is formally enacted. Tracking the changes between these versions can be difficult due to the volume and complexity of the documents, making it challenging for the public to verify whether the final provisions are consistent with those proposed or discussed during earlier stages. Consequently, the availability of tools that facilitate the comparison of legislative document versions can enhance public participation and strengthen civic oversight. This study proposes the application of a two-level Myers difference algorithm with middle snake optimization to identify differences between legislative documents. The first level performs line-based comparison to locate modified regions, while the second level performs character-based comparison to highlight detailed textual changes. The proposed approach aims to provide a clear visualization of revisions between document versions and support greater transparency throughout the legislative process.**

Keywords—legislative; version; myers; optimization; insert (key words)

I. Introduction

The formation of laws and regulations is an important process in democratic governance that requires transparency, accountability, and meaningful public participation. Throughout the legislative process, a bill may undergo multiple revisions before being formally enacted. These revisions can occur during drafting, deliberation, and amendment stages, resulting in several versions of the same document. Consequently, the ability of the public to oversee and understand changes made during the legislative process becomes an important aspect of civic participation.

However, tracking changes across successive versions of legislative documents is not always straightforward. Legislative documents are often lengthy and complex, making manual comparison inefficient and prone to error. Furthermore, differences between earlier drafts and enacted regulations may be difficult to identify without systematic comparison. This creates challenges for civil society, researchers, and other stakeholders seeking to verify whether provisions introduced, discussed, or proposed during previous stages are reflected in the final document. Therefore, mechanisms that facilitate the identification and visualization of changes between document versions are necessary to support transparency and informed public oversight.

Document differencing algorithms provide a means of identifying additions, deletions, and modifications between two versions of a text. One of the most widely used approaches is the Myers difference algorithm, which computes the shortest edit script between two sequences with a time complexity of O(ND), where N is the combined length of the sequences and D is the minimum edit distance. Moreover, the middle snake optimization proposed by Myers enables the algorithm to reduce memory consumption through a divide-and-conquer strategy while maintaining the same asymptotic time complexity.

This study proposes the application of a two-level Myers difference algorithm with middle snake optimization for comparing versions of legislative documents. At the first level, the algorithm performs line-based comparison to identify modified regions within the documents. Subsequently, character-level comparison is applied only to the modified lines to provide finer-grained visualization of textual changes. By limiting detailed comparisons to affected regions, the proposed approach seeks to provide clear and efficient identification of revisions between document versions.

The results of this study are expected to contribute to the development of tools that facilitate the comparison and visualization of legislative document revisions. Such tools may support greater transparency in the lawmaking process and enhance the ability of citizens and stakeholders to monitor changes throughout the evolution of legislative documents.

II. Theoretical Foundation

A. Legislative Documents in Indonesia

Legislation is defined as a written regulation containing generally binding legal norms that are established or enacted by authorized state institutions or officials through procedures prescribed by law. The process of legislative formation encompasses the stages of planning, drafting, deliberation,

enactment or stipulation, and promulgation. Law No. 13 of 2022 (UU No. 13 Tahun 2022), which constitutes the second amendment to Law No. 12 of 2011 (UU No. 12 Tahun 2011) concerning the Formation of Laws and Regulations, further emphasizes that the formation of legislation also includes monitoring and review activities. In addition, the law guarantees public access to information and opportunities to provide input, both orally and in writing, through online and offline mechanisms, throughout every stage of the legislative process.

The importance of public involvement in the legislative process was further strengthened by Constitutional Court Decision No. 91/PUU-XVIII/2020 (Putusan Mahkamah Konstitusi No. 91/PUU-XVIII/2020), which introduced the principle of meaningful participation. According to this principle, the public has the right to be informed about legislative activities, to express opinions and receive consideration of those opinions, and to obtain explanations or responses regarding the outcomes of public participation. These principles are intended to promote transparency, accountability, and democratic governance by ensuring that stakeholders and affected communities are able to actively participate in the formulation of laws and regulations.

In practice, however, legislative documents frequently undergo multiple revisions during the stages of drafting and deliberation before a regulation is formally enacted. The existence of numerous document versions, coupled with the length and complexity of legal texts, may make it difficult for citizens and other stakeholders to track modifications introduced throughout the legislative process. Identifying additions, deletions, or alterations between successive drafts and the enacted version often requires manual examination, which can be time-consuming and susceptible to oversight. Consequently, despite the availability of mechanisms for public participation, the ability of the public to effectively monitor the evolution of legislative documents may be hindered by the practical challenges associated with comparing different versions of legal texts.

Therefore, facilitating the identification and visualization of changes between legislative document versions is essential to support meaningful public participation and enhance transparency in the lawmaking process. Tools capable of systematically comparing successive versions of legislative documents can assist citizens, researchers, and civil society organizations in understanding how legal provisions evolve over time, thereby strengthening public oversight and promoting greater accountability in legislative decision-making.

B. Document Versioning and Difference Detection

Document versioning refers to the management and tracking of changes made to a document over time. In the context of legislative documents, multiple versions may exist throughout the drafting and deliberation process before a regulation is formally enacted. Consequently, identifying and understanding the differences between successive versions becomes an important aspect of ensuring transparency and facilitating public oversight.

Document difference detection aims to determine the modifications that transform one version of a document into another. Such modifications generally consist of insertions, deletions, and substitutions. Techniques for document comparison and change tracking are widely employed in version control systems to record revisions and present the differences between document versions in a comprehensible manner.

One of the fundamental concepts in document differencing is the shortest edit script (SES), which represents the minimum sequence of editing operations required to transform one sequence into another. Efficient computation of the shortest edit script enables the identification and visualization of changes while minimizing unnecessary edit operations. This concept forms the basis of numerous difference algorithms used in modern version control systems and text comparison tools.

C. Edit Graph

An edit graph is a graphical representation of the relationship between two sequences and provides a framework for interpreting sequence comparison as a path-finding problem. Given two sequences, one sequence is placed along the horizontal axis and the other along the vertical axis. Each node in the graph represents a pair of positions within the two sequences, while the edges correspond to possible editing operations required to transform one sequence into the other.

Three types of edges are commonly defined in an edit graph. A horizontal edge represents a deletion operation, indicating that an element from the first sequence is removed. A vertical edge represents an insertion operation, indicating that an element from the second sequence is added. In addition, when corresponding elements in both sequences are identical, a diagonal edge is formed, representing a match between the two elements and requiring no edit operation.

The process of transforming one sequence into another can therefore be interpreted as finding a path from the source node located at the upper-left corner of the graph to the destination node located at the lower-right corner. Among all possible paths, the shortest path corresponds to the minimum number of insertion and deletion operations required to transform one sequence into the other. Consequently, the edit graph establishes a direct relationship between sequence comparison and the shortest edit script problem.

D. Myers O(ND) Difference Algorithm

The Myers O(ND) Difference Algorithm, proposed by Myers in 1986, is an efficient method for computing the shortest edit script (SES) between two sequences. Unlike classical dynamic programming approaches that require quadratic time and memory, Myers' algorithm exploits the structure of the edit graph to determine the minimum sequence of insertion and deletion operations required to transform one sequence into another. The algorithm achieves a time complexity of O(ND), where N denotes the combined length of the two sequences and D represents the minimum edit distance between them.

A fundamental concept in Myers' algorithm is the D-path, which refers to a path in the edit graph containing exactly D non-diagonal edges corresponding to insertion and deletion operations. For each value of D, the algorithm determines the furthest-reaching path along every diagonal of the edit graph. A furthest-reaching path is defined as the path that reaches the greatest x-coordinate on a given diagonal using at most D edit operations. By progressively increasing D and extending these paths, the algorithm eventually discovers a path connecting the source node and the destination node, thereby obtaining the shortest edit script.

Another important concept introduced by Myers is the snake. A snake is a maximal sequence of consecutive diagonal edges in the edit graph, representing a contiguous region where the corresponding elements in both sequences are identical. Since diagonal movements correspond to matching elements and do not contribute to the edit distance, extending paths along snakes enables the algorithm to efficiently traverse unchanged regions of the sequences.

The shortest edit script obtained by Myers' algorithm describes the minimum set of insertion and deletion operations necessary to transform one sequence into another. Furthermore, the shortest edit script is closely related to the longest common subsequence problem, as both can be derived from the same shortest path in the edit graph. This relationship allows Myers' algorithm to efficiently determine sequence differences while avoiding the construction of the entire dynamic programming matrix.

Because the algorithm focuses on furthest-reaching paths rather than evaluating all possible states, it achieves a time complexity of O(ND), which is particularly advantageous when the two sequences are similar and the edit distance D is relatively small. Consequently, Myers' algorithm has become the foundation of many modern text comparison and version control systems due to its efficiency and ability to generate human-readable differences.

E. Divide and Conquer

Divide and conquer is a fundamental algorithmic paradigm that solves a problem by recursively decomposing it into smaller subproblems, solving each subproblem independently, and combining their results to form the final solution. This approach is particularly effective when subproblems are structurally similar to the original problem and can be solved independently without overlapping computations.

F. Middle Snake Optimization

The original Myers O(ND) Difference Algorithm computes the shortest edit script by exploring furthest-reaching paths in the edit graph. However, reconstructing the complete sequence of edit operations requires storing information for each iteration, resulting in memory consumption proportional to O(ND). To address this limitation, Myers proposed a divide-and-conquer optimization based on the concept of the middle snake, which reduces memory requirements while preserving the algorithm's time complexity.

The divide-and-conquer approach recursively partitions the edit graph into smaller subproblems. Instead of storing the entire search history, the algorithm simultaneously performs forward and reverse searches from the source and destination nodes, respectively. The forward search determines the furthest-reaching paths originating from the source node, while the reverse search computes furthest-reaching paths from the destination node. When the paths produced by the two searches intersect, the algorithm identifies a middle snake that lies on the shortest edit path.

A middle snake is defined as a maximal sequence of consecutive diagonal edges located near the midpoint of the shortest edit path. Since diagonal edges represent matching elements between the two sequences, the middle snake divides the edit graph into two independent subproblems. The algorithm then recursively applies the same procedure to the regions before and after the middle snake until the complete shortest edit script is obtained.

By employing forward and reverse searches together with the divide-and-conquer strategy, the middle snake optimization eliminates the need to store information for every D-path encountered during the search. Consequently, memory complexity is reduced from O(ND) to O(N), while the time complexity remains O(ND). This improvement enables efficient comparison of large sequences and forms the basis of many practical implementations of Myers' difference algorithm used in modern text comparison and version control systems.

III. METHODOLOGY

A. Two-Level Myers Algorithm

The proposed method employs a two-level application of the Myers difference algorithm to efficiently identify differences between two versions of a document. Instead of performing a character-level comparison on the entire document, the method first analyzes differences at the line level and subsequently applies character-level comparison only to the lines that have been identified as modified.

At the first level, the algorithm performs a line-by-line comparison between the two documents. Each line is treated as a single element in the sequence, allowing the algorithm to determine which lines have been inserted, deleted, or changed. This stage produces a set of candidate lines that require further examination.

At the second level, the algorithm applies the Myers difference algorithm at the character level only to the lines classified as modified during the first stage. This finer-grained comparison identifies the exact positions of inserted and deleted characters within those lines, providing detailed information about the changes while preserving the surrounding context.

The two-level approach offers several advantages over applying character-level Myers comparison directly to the entire document. When character-level comparison is performed on the whole document, every character in both documents becomes part of the input sequence. Consequently, the algorithm must process a significantly larger search space,

resulting in increased computational cost and memory usage. Furthermore, changes occurring in one part of the document may cause subsequent characters to shift, making the resulting differences more difficult to interpret.

By first performing comparison at the line level, unchanged lines are excluded from further processing, and character-level analysis is restricted only to regions where modifications actually occur. This reduces the amount of data processed during the second stage and improves overall efficiency. In addition, the resulting differences are more readable because modifications are localized within individual lines rather than being distributed across the entire document.

Therefore, the proposed two-level Myers algorithm combines the efficiency of coarse-grained line comparison with the precision of fine-grained character comparison, enabling accurate and computationally efficient detection of textual differences.

B. Common Prefix and Suffix Trimming

Before applying the Myers difference algorithm, an initial preprocessing step is performed to remove identical prefixes and suffixes shared by the two input documents. This step is based on the observation that common leading and trailing segments do not contribute to the difference computation, as they remain unchanged across both versions.

Let the two documents be represented as sequences A and B. The algorithm first compares elements from the beginning of both sequences. If A[i] = B[i] for consecutive positions, these elements are classified as a common prefix and excluded from further processing. Similarly, the algorithm compares elements from the end of both sequences. If A[n−j] = B[m−j] holds for consecutive positions, these elements are identified as a common suffix and are also excluded.

After removing the common prefix and suffix, the remaining subsequences A' and B' represent the minimal region that potentially contain differences. The Myers difference algorithm is then applied only to A' and B'. Once the shortest edit script is computed for the reduced sequences, the previously removed prefix and suffix are reattached to reconstruct the complete diff result.

This preprocessing step improves computational efficiency, especially when large portions of the documents remain unchanged at the boundaries. In such cases, prefix and suffix trimming reduces the effective input size, thereby decreasing the number of states explored by the Myers algorithm. Importantly, this optimization does not affect correctness, as it only removes guaranteed matching regions that do not contribute to the edit distance.

IV. IMPLEMENTATION

A. Implementation Environment

The program was implemented in Python as a command-line application. Python was selected because the main objective of this project is to demonstrate the algorithmic strategy used to compare two document versions, rather than to build a graphical interface or a production-scale document management system. The implementation only requires two input paths: the previous version of a legislative document and the latest version of the document.

The program reads both files using UTF-8 encoding and splits them into lines. Blank lines are preserved because spacing may be meaningful in legislative documents. The main execution flow is implemented in src/main.py, while file reading and command-line argument parsing are separated into src/io_utils.py.

B. Data Structures

The implementation uses two primary data structures. The first is DiffOp, which represents a single edit operation. Each operation has a kind and content. The kind field can be equal, insert, or delete, while the content field stores the compared line or character sequence.

The second structure is ModifiedLine, which represents a line that is considered modified rather than purely deleted or inserted. It stores the deleted line, the inserted line, and a list of character-level DiffOp objects. This separation allows the program to distinguish between line-level changes and character-level changes inside a modified line.

C. Modified-Line Pairing

Line-level Myers naturally reports changes as deletions and insertions. A modified sentence usually appears as one deleted line near one inserted line. Therefore, the implementation includes a pairing stage in src/line_diff.py.

The program scans each block of adjacent non-equal operations. Deleted and inserted lines inside the same block are compared using a similarity function. If the similarity score is at least 0.4, the pair is treated as a modified line.

The similarity function combines two measurements. The first measurement compares common prefix and suffix length. The second measurement compares character overlap using character frequency counts. The final score is the larger of the two. This allows the program to detect both ordinary sentence edits and repeated-character cases.

For blocks with multiple deleted and inserted lines, the implementation uses a small greedy local matching strategy. Candidate pairs are sorted by similarity score, then by distance inside the block. This helps pair the most likely modified lines while leaving unrelated deletions and insertions as separate operations.

D. Result Output

Here is an example of the program output:

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi resolution TIFF or EPS file with all fonts embedded) because this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord "Format" pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.

```
   UNDANG-UNDANG REPUBLIK INDONESIA
   NOMOR 10 TAHUN 2024
   TENTANG PENGELOLAAN DATA PUBLIK

   Pasal 1
*  Data publik adalah data yang dihasilkan oleh lembaga negara, badan publik, dan
 dapat diakses oleh masyarakat sesuai ketentuan peraturan perundang-undangan.

   Pasal 2
*  Setiap lembaga negara wajib menyediakan data publik secara akurat, lengkap, te
rbuka, dan tepat waktu.

   Pasal 3
*  Permohonan akses data publik dapat diajukan secara tertulis atau elektronik ke
pada pejabat pengelola informasi.

   Pasal 4
*  Lembaga negara dapat menolak permohonan data apabila data tersebut berkaitan d
engan rahasia negara atau data pribadi yang dilindungi.

   Pasal 5
*  MasyaraKekat benturhank memperolebih lpeanjut mengbenari tathuan capabrila per
mohonyediaan data publik diatur dengolank Peraturan Pemerintah.

   Pasal 6
+ Ketentuan lebih lanjut mengenai tata cara penyediaan data publik diatur dengan
Peraturan Pemerintah.
+
+ Pasal 7
   Undang-Undang ini mulai berlaku pada tanggal diundangkan.
```

VIDEO LINK AT YOUTUBE

The codebase of the implementation can be accessed in https://github.com/SlackingSlothh/LegislativeDocumentDiff.git

REFERENCES

[1] Republik Indonesia, Undang-Undang Nomor 12 Tahun 2011 tentang Pembentukan Peraturan Perundang-undangan, Lembaran Negara Republik Indonesia, 2011. Accessed: Jun. 19, 2026.

[2] Republik Indonesia, Undang-Undang Nomor 13 Tahun 2022 tentang Perubahan Kedua atas Undang-Undang Nomor 12 Tahun 2011, Lembaran Negara Republik Indonesia, 2022. Accessed: Jun. 19, 2026.

[3] Mahkamah Konstitusi Republik Indonesia, Putusan Nomor 91/PUU-XVIII/2020, 2020. Accessed: Jun. 19, 2026.

[4] E. W. Myers, "An O(ND) Difference Algorithm and Its Variations," Algorithmica, vol. 1, no. 2, pp. 251–266, 1986.

[5] S. Chacon and B. Straub, Pro Git, 2nd ed. Apress, 2014. (Git diff and version control literature reference)

[6] N. Fraser, "Diff Strategies," Google, 2006. Available: https://neil.fraser.name/writing/diff/ Accessed: Jun. 19, 2026.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd/4th ed. MIT Press.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Juni 2026

Ahmad Zaky Robbani, 13524045