
White-Box Basis Path Testing of the Bellman-Ford Algorithm in a Package Delivery Application

Abdi Pandu Kusuma^{1*}, Ayuna Rahim Imannur²

¹ Islamic University of Blitar, Faculty of Engineering and Informatics, Computer Systems, Blitar, Imam Bonjol Street No. 16, Jl. Majapahit No. 2–4, Sananwetan, Sananwetan District, Blitar City, East Java, 66137, Indonesia

² Indonesia Blitar Islamic University, Faculty of Engineering and Informatics, Informatics Engineering, Blitar, Imam Bonjol Street No. 16, Jl. Majapahit No. 2–4, Sananwetan, Sananwetan District, Blitar City, East Java, 66137, Indonesia

Article History

Submitted : 24-08-2026

Revised : 05-10-2026

Accepted : 05-10-2026

Published : 12-10-2026

DOI: 10.32664/smatika.v16i03.2576

Keyword

Basis Path Testing; Bellman-Ford Algorithm; Cyclomatic Complexity; Software Testing; White-Box Testing

*Corresponding Author:

pans.uib5blitar@gmail.com

Abstract

Route determination applications based on the Bellman-Ford algorithm are widely adopted because they handle weighted graphs iteratively, including negative weights. However, testing that only checks output correctness, such as black-box testing, cannot guarantee that every execution path, particularly inside the nested loops of the edge-relaxation process, behaves as intended, so hidden logical errors may remain undetected. This study verifies the structural correctness of the nearest delivery-distance module of a web-based package delivery application that implements the Bellman-Ford algorithm, using white-box testing with the basis path technique: flowgraph construction, cyclomatic complexity calculation, independent path determination, and the design and execution of test cases. The flowgraph consists of 15 nodes, 18 edges, and 4 decision nodes, giving a cyclomatic complexity $V(G)$ of 5, which is classified as simple and low-risk; five independent paths cumulatively cover all 18 edges. Executing the five path-based test cases, together with one supplementary test on a larger sample graph, produced outputs identical to the expected results in every case, so the implementation is structurally valid within the scope of the basis path coverage criterion. Nested loops do not by themselves raise $V(G)$: the number of independent paths depends on the number of decision points rather than on the number of iterations, so $V(G)$ should not be read as a proxy for the $O(|V||E|)$ running time of Bellman-Ford. Path-based testing additionally reached three paths, namely an empty edge set, negative-weight cycle detection, and an inner loop that is never entered, which the earlier black-box testing had not exercised. The five-path test set provides a replicable structural testing procedure for similar graph algorithm implementations, to be complemented by loop and boundary testing.

1. Introduction

The public's need for a fast and accurate route-determination system continues to grow alongside the rapid growth of location-based applications, whether for daily navigation, logistics distribution, or the search for the nearest public facilities. Shortest-path search algorithms, one of which is Bellman-Ford [1], [2], are widely adopted in the development of such applications because of their ability to handle weighted graphs iteratively, including cases involving negative weights that cannot be resolved by other algorithms such as Dijkstra. Various studies have demonstrated the application of Bellman-Ford in diverse contexts, ranging from public-

transportation route search toward campuses to Android-based navigation systems [3]. The widespread adoption of this algorithm in publicly accessed applications demands assurance that the program code implementing it is free of logical errors, since failures at the route-computation stage can directly affect the accuracy of the results received by users. This condition drives the urgency of structural software testing, rather than testing limited to the application's display or final output alone.

Although the accuracy of the final result is often used as a benchmark for the success of shortest-path determination applications, testing that focuses solely on output conformity has not been able to guarantee that the entire code execution path has run as intended. The iterative nature of the Bellman-Ford algorithm's structure, with nested loops performing edge relaxation $V-1$ times, has the potential to conceal hidden logical errors under certain conditions, for example, in handling graphs with cycles, isolated nodes, or loop boundary conditions that are rarely tested through ordinary functional testing. Black-box testing, which is commonly applied to web and mobile applications, tends to verify the conformity of input and output without tracing the program's internal logic path [4], [5], so that structural errors in the core part of the algorithm risk going undetected even though the application appears to function normally in common testing scenarios [6]. This is the main focus of the problem, namely, that the structural correctness of the program code in the implementation of the Bellman-Ford algorithm has not yet been fully verified.

A number of previous studies have applied white-box testing with the basis path technique to test the structural correctness of various applications. One study applying the technique to a login form obtained a low cyclomatic complexity value with all test cases declared valid [7], and further studies applied the technique to other login and registration forms [8]–[9], while another study obtained similar results in testing a web-based application with two independent paths [10]. A study applying basis path testing to an Android-based cash-flow application ensured that every execution path of the transaction module was tested at least once [11], while other studies applied the technique to a web-based library information system [12], a web-based booking form [13], and a promotion-search page [14]. The advantage of this approach lies in its ability to quantitatively measure the logical complexity of a program through the calculation of cyclomatic complexity and to ensure independent path coverage, as explained in its underlying framework [15], [16]. However, all of these studies were applied to application modules with relatively simple control structures, such as form validation and linear CRUD processes, so their limitations become apparent when confronted with program code containing nested loops and complex iterations. The closest precedent known to the authors applied basis path testing to pathfinding algorithms (iterative deepening search and Held-Karp) [17]; it did not address the edge-relaxation loop and the negative-cycle detection that characterize Bellman-Ford. A recent preprint further showed that automatic basis-path construction methods based on graph-traversal algorithms such as DFS/BFS are still prone to encountering dead ends (greedy traps) on complex control-flow graphs with intricate looping and branching structures, making it difficult to produce a complete set of basis paths [18]. Because that source is a preprint that, to the authors' knowledge, has not yet been peer reviewed, it is used here only as evidence that the problem is open, and peer-reviewed work on basis path coverage of complex programs likewise continues to rely on dedicated optimization-based test generation [19].

Based on the above description, it appears that research on white-box testing with the basis path technique has so far been centered on business-oriented applications with linear and simple branching control structures, whereas its application to program code implementing numerical-iterative algorithms such as Bellman-Ford, with nested loops between global relaxation iterations and the inspection of each graph edge, has not been widely explored. Because nested loops increase the number of times statements are executed, it is often assumed that they also increase cyclomatic complexity; whether this holds in practice is an open empirical question, since $V(G)$ counts decision points rather than iterations [15], [20]. Nested loops may also present distinct challenges in constructing flowgraphs and determining basis sets, related to the difficulty conventional methods face with complex control-flow graphs [18]. This research gap forms the basis of the novelty of this study, namely the application of white-box testing with the basis path technique specifically to the program code of the Bellman-Ford algorithm in a shortest-path determination application, encompassing the construction of a flowgraph from the nested looping structure, the calculation of cyclomatic complexity, the

determination of independent paths, and the design of test cases that specifically target the algorithmic characteristics of Bellman-Ford, rather than merely the interface module or business flow of the application as in previous studies.

Based on the research gap described above, this study aims to test the structural correctness of the program code in a shortest-path determination application that implements the Bellman-Ford algorithm through white-box testing with the basis path technique, encompassing flowgraph construction, cyclomatic complexity calculation, independent path determination, and the design and execution of test cases for each identified path. The study addresses three research questions: (RQ1) What are the flowgraph structure and the cyclomatic complexity of the nearest delivery-distance module that implements Bellman-Ford? (RQ2) How many independent basis paths does the module have, and can each of them be executed by a designed test case (path implementability)? (RQ3) Does white-box path tracing reveal logical anomalies or unexercised paths that the previously performed black-box testing did not? Theoretically, this study is expected to enrich the body of software-testing literature by extending the application of basis path testing techniques from the business-application domain, which has so far been dominant, to the domain of algorithmic-iterative program code, while providing empirical evidence regarding the path-complexity characteristics of the nested looping structure of a single algorithm, Bellman-Ford, in particular the relation between loop nesting and $V(G)$. Practically, the results of this study are expected to provide evidence of the structural correctness of the tested shortest-path determination application, minimize the risk of logical errors undetected by functional testing alone, and serve as a reference for structural testing procedures that can be replicated by other developers in testing similar graph algorithm implementations, such as Dijkstra or Floyd-Warshall, in similar applications (a cross-algorithm comparison is outside the scope of this study and is left for future work).

2. Research Method

This study was designed using a Research and Development (R&D) approach, chosen because it is oriented toward producing a new software application while scientifically testing its effectiveness [21], as applied in comparable information-system development studies [22]. Unlike descriptive or exploratory research, which stops at the needs-analysis stage, R&D requires a measurable cycle of design, development, and product verification so that the quality of the final result can be empirically accounted for [21]. The objective of this study is to verify the structural correctness of the nearest delivery-distance determination module of the developed package delivery application, particularly to detect errors that are not revealed through conventional functional testing [5]. The choice of R&D design is directly related to the research questions (RQ1–RQ3), namely:

- 1) The product development stage is a prerequisite for forming the test object, namely the Bellman-Ford algorithm module, whose structure is analyzed in terms of flowgraph structure and the cyclomatic complexity value of the program code (RQ1).
- 2) The product testing stage serves as the vehicle for answering questions regarding the number and implementability of basis paths (RQ2).
- 3) The effectiveness of white-box testing compared with black-box testing in uncovering logical anomalies (RQ3).

Thus, R&D was not simply chosen as an administrative framework for the research, but rather as the methodological foundation linking the system-development process with quality evaluation based on the cyclomatic-complexity metric first formulated by McCabe [15] and later standardized as a structured testing methodology by Watson and McCabe [16].

The data source for this study was obtained from a package delivery application operating at <https://jatinomindah.com/home>, which was selected as representative of a case study of a web-based package delivery application with a module for determining delivery distance and route estimation. The selection criteria for the application were based on the number of products available on the site, considering that the variation and volume of product data reflect the complexity of delivery scenarios that need to be verified for distance to various destination points. The data used in this study is product data available within the

application, which subsequently forms the basis for designing test scenarios for the nearest delivery-distance calculation module [13]. The approach of collecting data from an already operating application system is consistent with common practice in structural testing research, which places real web applications as test objects to ensure the relevance and replicability of test results in a production context [12]. Because all data used originates from product data and system data rather than personal user data, data collection at this stage did not involve interviews, surveys, or access to personal identity information, thereby minimizing risks to user privacy from the outset of the study.

The research framework consists of three sequential stages, namely system design, system development, and system testing. In the system-design stage, the logic flow of the nearest delivery-distance determination module, which implements the Bellman-Ford algorithm with a nested looping structure for the relaxation process of each edge of the delivery-route graph, was rendered into a flowchart as shown in Figure 1.

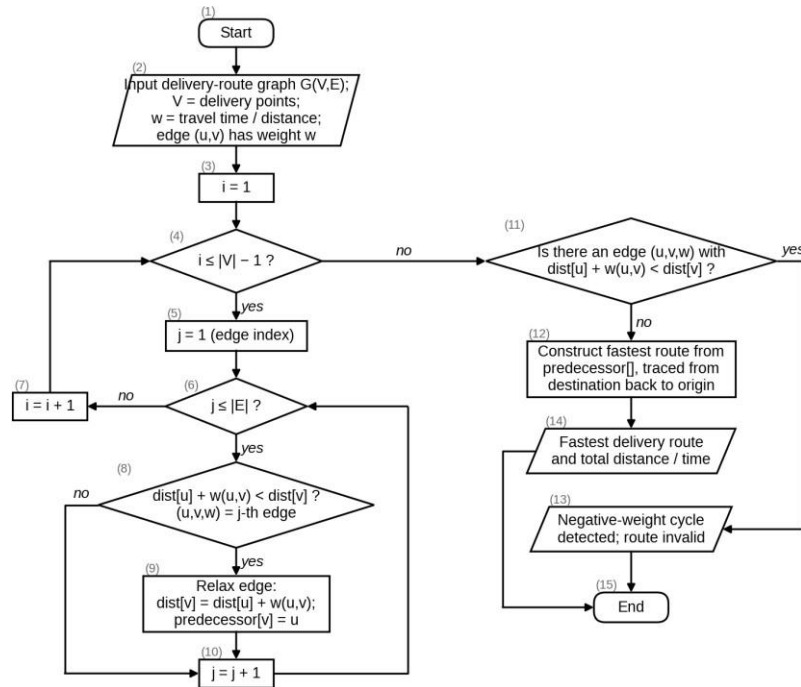


Figure 1. Bellman-Ford algorithm flowchart (node numbers in parentheses correspond to the flowgraph nodes in Figure 3).

This flowchart subsequently forms the basis for constructing the flowgraph in the testing stage, as is customary in path-based white-box testing procedures [15], [16]. In the system-development stage, the application was built using the PHP programming language with the Laravel framework and a MySQL database, a technology combination chosen for its ability to support a modular Model-View-Controller code structure, allowing the algorithm module to be tested in isolation without being affected by the user-interface layer, as demonstrated in a similar study that used Laravel as the development framework for a white-box test object [8]. The system-testing stage was carried out using the white-box testing technique with basis path testing, which includes constructing a flowgraph from the program code, calculating the cyclomatic complexity value using the formula $V(G) = E - N + 2p$, where E is the number of edges, N the number of nodes, and p the number of connected components ($p = 1$ for a single module), or equivalently $V(G) = \pi + 1$, where π is the number of predicate (decision) nodes [15], [16], determining the set of independent basis paths, and designing test cases that ensure each independent path is executed at least once [20]. The selection of basis path testing for the shortest-path search algorithm module is based on precedent research showing that this technique can uncover logical errors in pathfinding algorithms with looping and branching structures [17], making it relevant to the characteristics of the Bellman-Ford algorithm, which is the focus of RQ1 and RQ2. The flowgraph was derived from the

flowchart in Figure 1, which therefore defines the level of abstraction at which $V(G)$ is computed (see Section 3.8).

The experimental procedure was carried out by executing all the test cases that had been designed for the nearest delivery-distance calculation module, then comparing the program's actual output with the shortest-distance value theoretically produced by the Bellman-Ford algorithm [2], to ensure that the program logic conforms to the expected algorithmic specification. As a comparison model, the results of white-box testing using the basis path technique were compared with the results of functional (black-box) testing previously performed on the same module, with the aim of identifying logical errors that escaped detection by black-box testing but were revealed through tracing the code's execution path, in line with the reported complementary relationship between structural and functional testing [4], [5]. The evaluation metrics used in this study include the cyclomatic complexity value $V(G)$, the number of identified independent basis paths, the percentage of path implementability (path coverage) achieved through the designed test cases, and the application error-risk category determined based on the cyclomatic complexity threshold. The validity of the test results was examined through cross-checking between the number of regions in the flowgraph, the $V(G)$ value obtained from the graph-matrix calculation, and the number of decision nodes in the program code, so that consistency among these three calculation approaches serves as an indicator of the accuracy of basis path identification. The test result data were subsequently analyzed descriptively and quantitatively by determining the number of nodes, edges, cyclomatic complexity value, number of independent paths, and the execution-success status of each test case, to then be interpreted in answering the research problems related to path implementability and the application's error-risk level. From a research-ethics standpoint, all data used originates from application system data and product data and therefore does not involve personal user information; testing was carried out in a development environment without disrupting the operational service of the application for end users, so that this study poses no ethical risk to the privacy or continuity of service of the application under study.

3. Result and Discussions

3.1 Application and module under test

The software testing process using the white-box testing method based on the basis path testing technique was applied to the Bellman-Ford algorithm for finding the fastest delivery route on a weighted graph $G(V,E)$. The testing covers five stages: (1) flowgraph design, (2) calculation of the cyclomatic complexity value, (3) determination of independent paths, (4) design and execution of test cases, and (5) conclusions of the white-box testing results based on the relationship between cyclomatic complexity and the level of failure risk, followed by a comparison with the previous black-box testing.

Figure 2 shows the display of the application program for searching for the fastest package-delivery route. For the sample graph shown, with source point X, the module returned the route X-A-C-D-E-B with a total distance of 17.95 km and a travel time of 38 minutes.

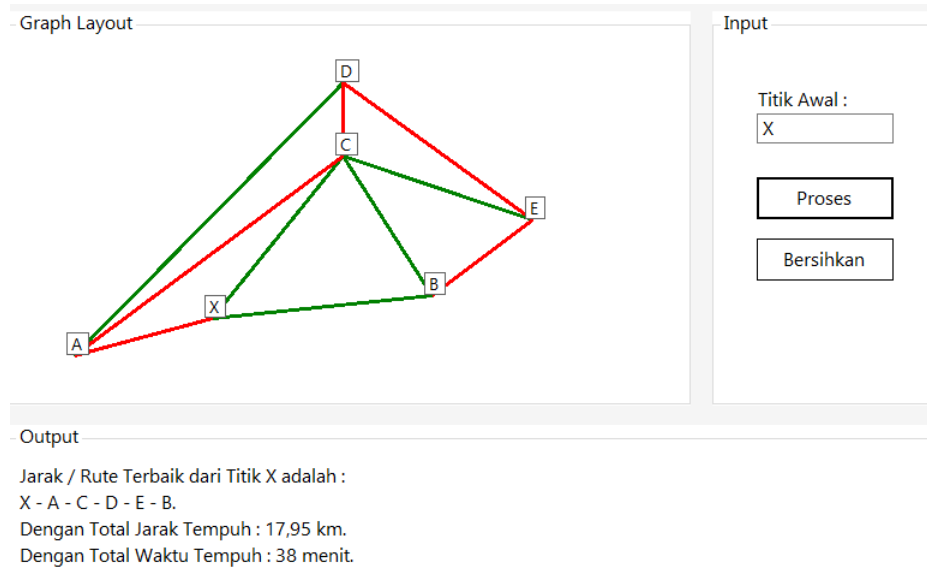


Figure 2. Display of the package delivery route search application.

3.2 Flowgraph construction

The flowchart of the package delivery application using the Bellman-Ford algorithm, which is the object of testing, has the following flow: it receives the input of the delivery-route graph $G(V,E)$ along with the weight w (travel time/distance), initializes the index i , then runs two nested loops to perform edge relaxation for each edge of the graph $|V|-1$ times, and concludes with a check for the presence of a negative-weight cycle before constructing and displaying the fastest delivery route.

Figure 3 below shows the resulting flowgraph, constructed based on Figure 1, representing the system's flow diagram using the Bellman-Ford algorithm. Table 1 defines the nodes and Table 2 lists the edges.

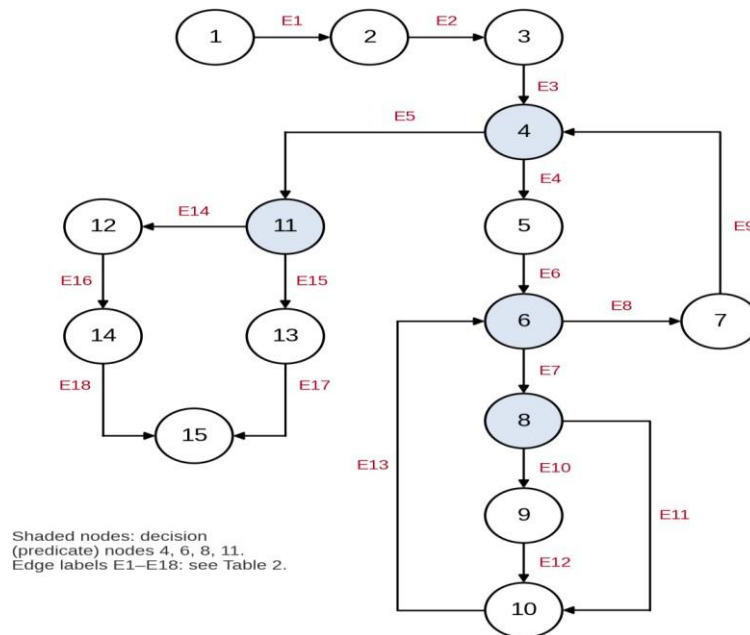


Figure 3. System application flowgraph using the Bellman-Ford algorithm.

Table 1. Node definitions of the flowgraph

Node	Statement in the module	Type
1	Start	Sequential
2	Input of route graph G(V,E) and weight w	Sequential
3	Initialization i = 1	Sequential
4	Outer-loop condition: $i \leq V - 1$	Decision (predicate)
5	Initialization j = 1	Sequential
6	Inner-loop condition: $j \leq E $	Decision (predicate)
7	$i = i + 1$	Sequential
8	Relaxation condition: $\text{dist}[u] + w(u,v) < \text{dist}[v]$	Decision (predicate)
9	Update: $\text{dist}[v] = \text{dist}[u] + w(u,v)$; predecessor[v] = u	Sequential
10	$j = j + 1$	Sequential
11	Negative-cycle check: is there an edge that can still be relaxed?	Decision (predicate)
12	Construct the fastest route from predecessor[]	Sequential
13	Output: negative-weight cycle detected; route invalid	Sequential
14	Output: fastest route and total distance/time	Sequential
15	End	Sequential

Based on Figure 3, the list of edges formed can be shown in Table 2. The total number of nodes (N) = 15, the total number of edges (E) = 18, with 4 decision nodes (predicate nodes), namely nodes 4, 6, 8, and 11.

Table 2. Edges formed from the flowgraph

No	Edge	Description
E1	1 → 2	Start to input of route graph G(V,E) and weight w
E2	2 → 3	Input to initialization of i = 1
E3	3 → 4	Initialization of i to checking the outer loop condition
E4	4 → 5 (yes)	Outer loop continues → initialization of j = 1
E5	4 → 11 (no)	Outer loop ends → proceeds to negative-cycle check
E6	5 → 6	Initialization of j to checking the inner loop condition
E7	6 → 8 (yes)	Inner loop continues → test the relaxation condition of edge j
E8	6 → 7 (no)	Inner loop ends → $i = i + 1$
E9	7 → 4	Return to checking the outer loop condition
E10	8 → 9 (yes)	Relaxation condition met → update distance & predecessor
E11	8 → 10 (no)	Relaxation condition not met → proceed directly to $j = j + 1$
E12	9 → 10	After the update → $j = j + 1$
E13	10 → 6	Return to checking the inner loop condition
E14	11 → 12 (no)	No edge can be relaxed → construct the fastest route
E15	11 → 13 (yes)	An edge can still be relaxed → negative cycle detected
E16	12 → 14	Route constructed → display fastest route & total distance
E17	13 → 15	Invalid output leads to End
E18	14 → 15	Fastest-route output leads to End

3.3 Cyclomatic complexity

Testing of the application system using White Box Testing was performed by determining the Cyclomatic Complexity value $V(G)$, where $V(G)$ measures the logical complexity of a program based on the flowgraph while also determining the minimum number of independent paths (basis paths) that must be tested. The $V(G)$ value is calculated using the equation:

$$V(G) = E - N + 2p \quad (1)$$

where $p = 1$ is the number of connected components. Based on the flowgraph in Figure 3, the number of edges

(E) obtained is 18 and the number of nodes (N) is 15. Thus, the $V(G)$ value obtained is:

$$V(G) = 18 - 15 + 2 = 5 \quad (2)$$

Following the validation procedure described in Section 2, this value was cross-checked in three further ways. First, the flowgraph has five regions: four bounded regions (the outer loop, the inner loop, the relaxation selection, and the final negative-cycle selection) plus the exterior region. Second, the graph (connection) matrix gives the same result: nodes 4, 6, 8, and 11 each have two outgoing edges, so $\Sigma(\text{outgoing edges} - 1) + 1 = 4 \times 1 + 1 = 5$. Third, the number of predicate nodes gives $V(G) = \pi + 1 = 4 + 1 = 5$. All approaches agree, which supports the accuracy of the flowgraph and of the identification of the basis paths.

The white-box testing results in this document are based on the relationship between the cyclomatic complexity value $V(G)$ and the level of software failure risk. Table 3 shows the risk-level classification based on the commonly used $V(G)$ value ranges as a reference [15], [16].

Table 3. Risk-level classification based on the cyclomatic complexity value range

V(G) value	Complexity category	Failure risk level
1 – 10	Simple (a simple program)	Low
11 – 20	More complex	Moderate
21 – 50	Complex (high risk)	High
> 50	Very complex / cannot be properly tested (untestable)	Very high

Based on the $V(G)$ calculation results, testing of the algorithm in the package delivery route system using the Bellman-Ford algorithm obtained a $V(G)$ value of 5, which falls within the range of 1–10. Referring to Table 3, the following conclusions can be drawn:

- 1) The logical complexity of the algorithm is classified as simple, with a $V(G)$ value of 5, despite having a nested looping structure and 4 decision points, so the failure risk resulting from control-structure complexity is classified as Low.
- 2) The program remains within a complexity range that is easy to understand, trace, and maintain, and does not require further simplification of the logic structure before testing proceeds.
- 3) To ensure that every execution path of the algorithm has been tested at least once, at least 5 test cases representing Path 1 through Path 5 must be created and executed, as designed in Table 4 and reported in Section 3.5, including scenarios of successful edge relaxation, failed edge relaxation, and negative-weight cycle detection.

3.4 Independent paths

In determining the independent paths, since this flowgraph contains 4 decision nodes (nodes 4, 6, 8, and 11), the minimum number of independent paths that must be formed is 5 paths (in accordance with the $V(G)$ value calculated above). These five paths were designed so that, collectively, they cover all 18 edges of the flowgraph, as shown in Table 4. Each path introduces at least one edge not used by the preceding paths: Path 1 covers E1–E3, E5, E14, E16, and E18 (7 edges); Path 2 adds E15 and E17; Path 3 adds E4, E6, E8, and E9; Path 4 adds E7, E11, and E13; and Path 5 adds E10 and E12. This confirms that the paths are linearly independent and that $7 + 2 + 4 + 3 + 2 = 18$ edges are covered.

Table 4. Determination of independent paths

Path	Node sequence	Scenario description
Path 1	1-2-3-4-11-12-14-15	The outer loop stops immediately ($ V -1 = 0$) and no edge can be relaxed, so the fastest route is constructed directly and is valid.
Path 2	1-2-3-4-11-13-15	The outer loop stops immediately, but the final check still finds an edge that can be relaxed, so a negative-weight cycle is detected and the route is invalid.
Path 3	1-2-3-4-5-6-7-4-11-12-14-15	The outer loop runs one iteration, but the inner loop stops immediately ($E = 0$ edges processed), so it proceeds to the next iteration of i before the final check.

Path	Node sequence	Scenario description
Path 4	1-2-3-4-5-6-8-10-6-7-4-11-12-14-15	The inner loop processes one edge whose relaxation condition is not met (no distance update), then proceeds to the next iteration.
Path 5	1-2-3-4-5-6-8-9-10-6-7-4-11-12-14-15	The inner loop processes one edge whose relaxation condition is met, so that $\text{dist}[v]$ and $\text{predecessor}[v]$ are updated, then proceeds to the next iteration.

3.5 Test-case design and execution

To answer RQ2, one test case was designed for each basis path (Table 5). The inputs are the smallest weighted graphs that force the module through the intended path, and the expected outputs were derived by hand from the Bellman-Ford specification [2] before execution. Inputs are given to the module as directed edge lists; if the module expands an undirected edge into two directed edges, the input must be adapted so that only the intended edge is processed. TC2 requires a negative edge weight, which the application does not normally produce for distance or travel time; it was therefore executed directly against the module (unit level) rather than through the user interface. TC6 is a supplementary test on the sample graph of Figure 2, added to observe repeated loop iterations on a realistic graph. The path actually traversed in each test was verified by PHPUnit with trace logging.

Table 5. Test cases for the five basis paths and their execution results

TC	Path	Scenario and input	Expected result	Actual result	Status
TC1	P1	One vertex (source X), no edges: $ V = 1$, $ E = 0$.	Outer loop not entered ($ V - 1 = 0$); no relaxable edge; route = X, total distance 0 km.	Outer loop not entered ($ V - 1 = 0$); no relaxable edge; route = X, total distance 0 km.	Pass
TC2	P2	One vertex X with a self-loop (X,X) of weight -1 (negative).	Outer loop not entered; final check finds $0 + (-1) < 0$; output “negative-weight cycle detected; route invalid”.	Outer loop not entered; final check finds $0 + (-1) < 0$; output “negative-weight cycle detected; route invalid”.	Pass
TC3	P3	Two vertices X (source) and A, no edges: $ V = 2$, $ E = 0$.	Outer loop runs once, inner loop not entered; $\text{dist}[X] = 0$, $\text{dist}[A] = \infty$; no negative cycle; A reported unreachable.	Outer loop runs once, inner loop not entered; $\text{dist}[X] = 0$, $\text{dist}[A] = \infty$; no negative cycle; A reported unreachable.	Pass
TC4	P4	Two vertices X (source) and A; one directed edge (A,X) of weight 5.	Relaxation test $\infty + 5 < 0$ is false; no update; $\text{dist}[X] = 0$, $\text{dist}[A] = \infty$; no negative cycle.	Relaxation test $\infty + 5 < 0$ is false; no update; $\text{dist}[X] = 0$, $\text{dist}[A] = \infty$; no negative cycle.	Pass
TC5	P5	Two vertices X (source) and A; one directed edge (X,A) of weight 4 km.	Relaxation test $0 + 4 < \infty$ is true; $\text{dist}[A] = 4$, $\text{predecessor}[A] = X$; final check finds no relaxable edge; route $X \rightarrow A$, 4 km.	Relaxation test $0 + 4 < \infty$ is true; $\text{dist}[A] = 4$, $\text{predecessor}[A] = X$; final check finds no relaxable edge; route $X \rightarrow A$, 4 km.	Pass
TC6	P4, P5	Supplementary (repeated loop iterations): sample graph of Figure 2 (six vertices X, A–E; source X).	X–A–C–D–E–B; 17.95 km; 38 min (recomputed precisely from the edge weights; see Table 6 and the note below).	X–A–C–D–E–B; 17.95 km; 38 min.	Pass

The expected value for TC6 was first derived by rounding the edge weights of the sample graph to the nearest whole kilometre and minute, which gave 18 km and 40 min; this rounded estimate did not match the actual output and was initially recorded as a failure. Recomputing the expected value precisely, without rounding the intermediate edge weights, gives 17.95 km and 38 min, which is identical to the actual output; TC6 is therefore corrected to Pass and the discrepancy was a rounding error in the hand-derived expected value, not a defect in the module.

Because the expected value of TC6 was revised after the test was executed, readers must be able to verify it independently. Table 6 therefore reports the distance and the travel-time weight of every edge of the sample

graph in Figure 2 (the ten edges drawn in the figure), and the total along the route X-A-C-D-E-B is obtained by adding the weights of its five edges. The five edges outside the route are listed for completeness, so that the route can also be compared with the alternative routes of the graph.

Table 6. Distance and travel-time weight of every edge of the sample graph in Figure 2

Edge	Endpoints	Distance (km)	Travel time (min)	On route X-A-C-D-E-B
1	X – A	1,29	2	Yes
2	A – C	2,20	7	Yes
3	C – D	2,35	8	Yes
4	D – E	2,11	3	Yes
5	E – B	2,04	3	Yes
6	A – D	0,92	1	No
7	X – C	2,18	4	No
8	X – B	1,76	3	No
9	C – B	1,47	3	No
10	C – E	1,63	4	No
Total	Route X-A-C-D-E-B (edges 1-5)	17.95	38	—

Path implementability, defined as the number of basis paths successfully executed divided by the total number of basis paths, was 5 of 5. The supplementary test TC6 also matched its (corrected) expected output, so no logical anomaly was found in any of the six test cases.

3.6 Comparison with black-box testing

To answer RQ3, the results were compared with the functional (black-box) testing previously performed on the same module, which consisted of six equivalence-partitioning test cases run on delivery-route graphs of five to six vertices with positive edge weights. The six cases are listed in Table 7 with their partition, graph size, and expected result; the test log predates this study and is held internally by the development team rather than published.

Table 7. Equivalence-partitioning test cases of the earlier black-box testing

Test case	Equivalence partition	Graph size	Expected result
BB1	single direct route	6 vertices	Route X-A-C-D-E-B; the total distance and time are equal to the weight of edge X-A-C-D-E-B.
BB2	route requiring several relaxations	6 vertices	Route X-A-C-D-E-B; the total distance equals the weight of edge X-A-C-D-E-B; however, the resulting time tends to be excessive compared to BB1.
BB3	unreachable destination	6 vertices	Route X-A-C-D-E-B; the total distance and time cannot be determined based on the edge weights of X-A-C-D-E-B.
BB4	single direct route with different direction	6 vertices	Route X-A-C-D-E-B; the total distance and time are equal to the weight of edge X-A-C-D-E-B.
BB5	route requiring more relaxations	6 vertices	Route X-A-C-D-E-B; the total distance equals the weight of edge X-A-C-D-E-B; however, the resulting time tends to be excessive compared to BB2.
BB6	reachable destination	6 vertices	Route X-A-C-D-E-B; the total distance and time are equal to the weight of edge X-A-C-D-E-B.

Table 8 shows, for each basis path, whether that black-box test set reached it. Paths 1 to 3 correspond to

degenerate inputs (a graph with no edges) or a negative-weight cycle, neither of which appears in the realistic delivery graphs used for functional testing. The “Yes” entries for Paths 4 and 5 in Table 8 are taken from the test log itself, which records that these two paths were exercised; they are not inferred from the type of input. White-box testing therefore exercised three paths that the earlier black-box testing had not, but did not reveal a logical anomaly in any of them: all three produced the expected, structurally valid behavior.

Table 8. Comparison between the previous black-box testing and the white-box basis path testing

Path	Scenario	Reached by black-box tests?	Reached by white-box test	Outcome
P1	Graph without edges (single vertex)	No	Yes (TC1)	Not previously tested
P2	Negative-weight cycle detected	No	Yes (TC2)	Not previously tested
P3	Outer loop runs, inner loop not entered	No	Yes (TC3)	Not previously tested
P4	Edge examined, no relaxation	Yes	Yes (TC4, TC6)	Previously tested
P5	Edge examined, relaxation performed	Yes	Yes (TC5, TC6)	Previously tested
Total	Basis paths reached; logical anomalies found	2 of 5	5 of 5	0 anomalies

3.7 Discussion

The present results agree with earlier basis path studies on the outcome, but not with what the nested loops of the algorithm suggested at the outset. Studies on login and registration forms and other web modules report low cyclomatic complexity and valid paths [7], [9], [10]; a registration form, for example, yielded four independent paths [9] and another web application two [10]. The five independent paths found here for Bellman-Ford are only slightly more, and the module falls into the same low-risk band (Table 3). The Introduction raised, as an open empirical question, whether nested loops would produce a much higher $V(G)$ and number of independent paths than in these conventional modules.

The discrepancy is explained by what $V(G)$ measures. Cyclomatic complexity equals the number of predicate nodes plus one [15], [16], so each loop contributes one decision regardless of how many times it iterates or how deeply it is nested. The repeated execution of the relaxation step, which gives Bellman-Ford its $O(|V||E|)$ running time [2], is therefore invisible to $V(G)$. Two factors together produce the low value: the module is small, with only four predicates (outer loop, inner loop, relaxation test, and final negative-cycle check), and $V(G)$ is a structural metric of control flow rather than a measure of algorithmic or numerical complexity [20], [23]. The low-risk classification in Table 3 must consequently be read as a statement about control-flow structure only.

This also delimits what a basis path set can show. The five paths exercise each loop with zero or one iteration and cover every edge, but they do not exercise behavior that emerges over many iterations, such as a distance that becomes final only after several passes, or the interaction between edge order and the number of passes required. Basis path testing is therefore necessary but not sufficient for iterative algorithms and should be complemented by loop and boundary-value tests and by stronger coverage criteria such as prime path coverage [24]. Path 2 raises a related point: it is feasible at module level but cannot be reached through the user interface as long as weights are non-negative distances or times, which illustrates the infeasible-path problem of structural testing [24]; the path remains relevant if the module is reused with other weight types, such as costs that may include discounts.

Regarding the difficulty of constructing basis sets reported in recent work [18], the five paths here could be constructed manually and verified to be independent and complete (Section 3.4). This is consistent with the observation that such difficulties concern larger and more intricate control-flow graphs: in that work the DFS/BFS baseline failed on 5 of 50 real-code functions with cyclomatic complexity between 1 and 8 that contained specific loop structures, whereas the graph here has $V(G) = 5$. Whether an automated DFS/BFS construction would have succeeded on this particular flowgraph was not evaluated in this study.

3.8 Limitations

This study has several limitations. First, the flowgraph was derived from the flowchart of the module (Figure

1), which represents the final negative-cycle check as a single predicate (“is there an edge that can still be relaxed?”) and treats the initialization of the distance array as sequential steps. If the source code implements these parts as loops or with compound conditions (for example, skipping edges whose source distance is infinite), each additional predicate would raise $V(G)$ by one and the code-level value would exceed 5; the reported $V(G)$ should therefore be read as the complexity at the abstraction level of Figure 1 rather than as a value verified directly against the source code. Second, only one module of one application was analyzed, so the results cannot be generalized to other implementations of Bellman-Ford. Third, the expected outputs were derived by hand from the algorithm specification, and the black-box comparison relies on a test set prepared earlier. Fourth, the risk categories in Table 3 are general thresholds and were not calibrated against defect data for this application.

4. Conclusions and Future Works

This study successfully applied white-box testing with the basis path testing technique to verify the structural correctness of the Bellman-Ford algorithm’s program code in a nearest delivery-distance determination application system, thereby addressing the research gap concerning the limited application of this technique to algorithmic-iterative program code with nested looping structures. Based on the flowgraph constructed from the program’s logic flow, 15 nodes, 18 edges, and 4 decision nodes (predicate nodes), namely nodes 4, 6, 8, and 11, were obtained, which subsequently yielded a cyclomatic complexity value $V(G)$ of 5 through the calculation $V(G) = E - N + 2p$, confirmed by the region count, the graph matrix, and the number of predicate nodes. This value falls within the range of 1–10, so the program’s logical complexity is classified as simple with a Low failure risk. A total of five independent paths were successfully identified and constructed so that, collectively, they cover all 18 edges of the flowgraph. Executing the five path-based test cases, plus one supplementary test, produced actual outputs identical to the expected results in all six cases (Table 5); within the scope of the basis path coverage criterion, the implementation of the Bellman-Ford algorithm in this delivery route determination application is therefore considered structurally valid. Path-based testing additionally reached Paths 1 to 3, which the earlier black-box scenarios had not exercised (Table 8), and revealed no logical anomaly in any of the five paths. The novelty of the study is a replicable five-path structural test procedure that targets the edge-relaxation loops and the negative-cycle check of Bellman-Ford, a structure that earlier basis path studies of form validation and CRUD modules did not address.

The findings have two implications. Theoretically, they show that the nested-loop structure of Bellman-Ford does not by itself raise cyclomatic complexity: $V(G)$ counts decision points rather than iterations, so it should not be used as a proxy for the algorithmic complexity $O(|V||E|)$ or as the sole risk indicator for numerical-iterative code, and basis path testing secures structural coverage of the loop boundaries but must be complemented by loop, boundary-value, and other structural criteria to address behavior that emerges over many iterations. Practically, the five test cases (empty edge set, negative-cycle detection, inner loop not entered, edge examined without relaxation, and edge relaxed) form a minimal, replicable structural test set for other graph algorithm implementations that share the same edge-relaxation pattern.

For future research, it is recommended that this basis path testing technique be replicated on implementations of other shortest-path search algorithms, such as Dijkstra and Floyd–Warshall, so that it can be used to compare path-complexity characteristics across algorithms, which was not done in the present study. Because $V(G)$ is a property of the code rather than of the input data, testing on delivery route graphs with substantially larger numbers of nodes and edges should target behavior that depends on the number of loop iterations, and performance, through loop testing and boundary-value analysis, rather than the stability of $V(G)$. Future research should also explore the development of automated tools for flowgraph construction and basis path determination on program code with complex control structures, given that conventional graph-traversal-based methods remain prone to greedy traps in cases of intricate looping and branching. In addition, the integration of basis path testing with other structural testing techniques, such as mutation testing or condition coverage, can be further examined to strengthen quality assurance for numerical-iterative algorithm modules.

5. Declarations

Author Contributions

Author 1 is responsible for Conceptualization, Methodology, Investigation, and Writing papers. Author 2 is responsible for Writing papers.

Funding

This research received no external funding.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

The flowgraph, the test cases, and their results supporting the white-box findings are reported in this article (Tables 1–2 and 5). The operational data from Jatinomindah supporting this research are not publicly available due to enterprise privacy.

Use of Artificial Intelligence (AI)

AI was used as a supporting tool during the preparation of this manuscript, particularly to assist with sentence structure improvement, grammar correction, academic language refinement, manuscript organization, and translation. The authors reviewed all content and take full responsibility for the manuscript; no AI tool is listed as an author.

Ethical Clearance

This study did not involve human participants, personal data, or animals requiring ethical approval. Ethical approval was therefore not required.

6. References

- [1] R. Bellman, "On a routing problem," *Q. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958, doi: 10.1090/qam/102435.
- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
- [3] S. Farhan, S. Andryana, and N. Hayati, "Implementasi Bellman-Ford dan Floyd-Warshall dalam Menentukan Jalur Terpendek Menuju Universitas Nasional Berbasis Android," *JUPI (J. Ilm. Penelit. dan Pembelajaran Inform.)*, vol. 5, no. 2, pp. 123–132, 2020, doi: 10.29100/jipi.v5i2.1812.
- [4] D. Wintana, D. Pribadi, and M. Y. Nurhadi, "Analisis Perbandingan Efektifitas White-Box Testing dan Black-Box Testing," *J. Larik Ldng. Artik. Ilmu Komput.*, vol. 2, no. 1, pp. 8–16, Jul. 2022, doi: 10.31294/larik.v2i1.1382.
- [5] H. Nurfauziah and I. Jamaliyah, "Perbandingan Metode Testing antara Blackbox dengan Whitebox pada Sebuah Sistem Informasi," *J. Vis.*, vol. 8, no. 2, pp. 105–113, Oct. 2022.
- [6] M. E. Khan, "Different approaches to white box testing technique for finding errors," *Int. J. Softw. Eng. Its Appl.*, vol. 5, no. 3, pp. 1–14, Jul. 2011.
- [7] M. F. Londjo, "Implementasi White Box Testing dengan Teknik Basis Path pada Pengujian Form Login," *J. Siliwangi Seri Sains dan Teknol.*, vol. 7, no. 2, pp. 35–40, 2021, doi: 10.37058/jssainstek.v7i2.4086.
- [8] M. I. Shiddiq, "Implementasi White Box Testing Berbasis Path pada Form Login Aplikasi Berbasis Web," *J. Siliwangi Seri Sains dan Teknol.*, vol. 8, no. 1, pp. 1–6, 2022, [Online]. Available: <https://doi.org/10.37058/jssainstek.v8i2.4098>
- [9] Y. J. Solissa, F. Putra, A. N. Putri, and S. R. C. Nursari, "Pengujian White Box Berbasis Path pada Form Daftar Jobstreet.co.id," *KONSTELASI Konvergensi Teknol. dan Sist. Inf.*, vol. 3, no. 2, pp. 353–362, Dec. 2023, doi: 10.24002/konstelasi.v3i2.8287.
- [10] M. G. A. Khamaeni, "Implementasi White Box Testing Berbasis Path pada Aplikasi Berbasis Web," *J. Siliwangi Seri Sains dan Teknol.*, vol. 9, no. 1, pp. 8–13, 2023, doi: 10.37058/jssainstek.v9i1.4109.
- [11] C. T. Pratata, E. M. Asyer, I. Prayudi, and A. Saifudin, "Pengujian White Box pada Aplikasi Cash Flow Berbasis Android Menggunakan Teknik Basis Path," *J. Inform. Univ. Pamulang*, vol. 5, no. 2, p. 111, 2020, doi: 10.32493/informatika.v5i2.4713.
- [12] R. I. Ndaumanu, "Pengujian Sistem Informasi Perpustakaan Berbasis Website dengan Basis Path Testing," *Justek J. Sains dan Teknol.*, vol. 6, no. 1, p. 123, 2023, doi: 10.31764/justek.v6i1.13808.
- [13] J. B. Sie, I. A. Musdar, and S. Bahri, "Pengujian White Box Testing terhadap Website Room Menggunakan Teknik Basis Path," *KHARISMA Tech*, vol. 17, no. 2, pp. 45–57, 2022, doi: 10.55645/kharismatech.v17i2.235.

- [14] Y. Pratiwi and L. W. Widiarti, "Implementasi Whitebox Testing Dengan Teknik Basis Path Pada Pengujian Halaman Pencarian Program Promo," *J. Kecerdasan Buatan dan Teknol. Inf.*, vol. 4, no. 2, pp. 173–180, May 2025, doi: 10.69916/jkbti.v4i2.280.
- [15] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.
- [16] A. H. Watson and T. J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," Gaithersburg, MD, USA, 1996. doi: 10.6028/NIST.SP.500-235.
- [17] I. G. S. Rahayuda and N. P. L. Santiari, "Basis path testing of iterative deepening search and Held-Karp on pathfinding algorithm," *Kursor*, vol. 9, no. 2, pp. 39–48, 2018, [Online]. Available: <https://doi.org/10.28961/kursor.v9i2.129>
- [18] C. Wei, X. Peng, Y. Yan, M. Luo, and T. Cai, "Rethinking basis path testing: Mixed integer programming approach for test path set generation," Jan. 2026. doi: 10.48550/arXiv.2601.05463.
- [19] B. Sun, L. Gong, Y. Guo, and D. Gong, "Basis path coverage testing of MPI programs based on multi-task evolutionary optimization," *Expert Syst. Appl.*, vol. 255, p. 124557, 2024, doi: 10.1016/j.eswa.2024.124557.
- [20] C. Ebert, J. Cain, G. Antoniol, S. Counsell, and P. Laplante, "Cyclomatic complexity," *IEEE Softw.*, vol. 33, no. 6, pp. 27–29, Nov. 2016, doi: 10.1109/MS.2016.147.
- [21] Sugiyono, *Metode Penelitian dan Pengembangan (R\&D)*. Bandung, Indonesia: Alfabeta, 2019.
- [22] D. E. Tawape, A. Y. Dako, R. D. R. Dako, W. Musa, W. Ridwan, and I. Z. Nasibu, "Research and Development (RnD) pada pengembangan sistem informasi manajemen PKK Desa Dunggala Kecamatan Tibawa Kabupaten Gorontalo," *J. Tek. Elektro dan Komput. TRIAC*, vol. 10, no. 2, 2023, [Online]. Available: <https://doi.org/10.21107/triac.v10i2.20542>
- [23] M. Shepperd, "A critique of cyclomatic complexity as a software metric," *Softw. Eng. J.*, vol. 3, no. 2, pp. 30–36, Mar. 1988, doi: 10.1049/sej.1988.0003.
- [24] P. Ammann and J. Offutt, *Introduction to Software Testing*, 2nd ed. Cambridge, U.K.: Cambridge University Press, 2016. [Online]. Available: <https://doi.org/10.1017/9781316771273>