**Introduction**

Much of today’s computer network optimization techniques, such as the distance-vector routing protocol used in packet switching, are derived from early solutions to the combinatorial optimization problems. The underlying techniques used in packet switching protocols come from the Bellman Ford algorithm, which was first proposed by Alfonso Shimbel, in his attempt to solve the shortest path problem. In technical terms, the shortest path problem involves finding a path that connects two nodes on a graph, such that the sum of the edge weights is minimized. In respect to computer networks, the aforementioned *nodes* represent the origin and destination files, while *edge weights* represent the costs associated with routing (e.g. distance and time). Answering the shortest path problem is pertinent to the study of packet switching because it provides the algorithm needed to facilitate the flow of packets between source and destination. According to Schrijver, the shortest path problem is different from other combinatorial optimization problems, such as spanning tree and transportation, in that actual mathematical research on this topic did not start until the 1950s (Schrijver, 2010, 155). Additionally, the algorithmic solution to the shortest path problem is relevant for its broader industrial applicability and influence on other academic fields beyond the scope of computer science.

The seminal paper that first proposed the solution to the shortest path problem was Alfonso Shimbel’s 1954 paper titled “Structure in Communication Nets”. Shimbel provided a novel approach to solving the shortest path problem. First, he differed from prior academics in *how* he viewed combinatorial optimization problems. Existing algorithms such as the Trémaux Algorithm, which used a depth-first search technique, along with Rosenfeld’s heuristical approaches would not have been sufficient to solve the shortest path problem, since the vulnerabilities of their approaches lie in the presence of (1) perpetual loops and (2) the tendency to calculate non-minimal solutions (Schrijver, 2010, 155). Simply put, existing techniques were capable of finding short paths but not the *shortest* path.

To find the shortest path, Shimbel used the matrix product method, which was largely inspired by his prior research in using matrices to model communication structures in neural networks. The matrix method translates data points from the graphical nets into a square matrix structure, which then allowed Shimbel to more effectively select the lowest-cost path from all sets of reachable paths between two nodes. Furthermore, Shimbel explains that the matrices can take on either symmetric or asymmetric forms depending on the presence of intermediary nodes. The actual matrix method is explained in the following section.

**Classic paper proposal**

In the matrix method, the graphical representation of finite nets is translated into a matrix form. For example, if there are n nodes in the graph, then the matrix will be n by n. Each element in the matrix, located in the ith row and jth column, is equal to the length of the *direct* route from node i to node j. In the case that a direct path does not exist between two nodes, the element will take on a value of infinity. The calculated distance between a node and itself is 0. The aforementioned matrix structure takes on a symmetric form. The symmetric matrix represents a basic network structure and is not as robust. This is because the symmetric matrix only considers values of direct paths, and even if there is an indirect but shorter path between two nodes, the symmetric matrix will not account for it. However, its counterpart, the asymmetric matrix, will incorporate the values of non-direct paths when finding the shortest path.

To ensure that his matrix method accounts for indirect routes, Shimbel introduced the following minimum-sum algebra.

For any arbitrary real or infinite numbers x and y, *x* + *y* = min (*x*,*y*) and *xy* = the algebraic sum of *x* and *y*; also ∞ · *x* = *x* · ∞ = ∞ for all *x* including ∞ (Shimbel, 1954, 199).

In the case that new links are added into a communication net to improve efficiency, then the new square matrix being formed is a product of square matrix A and square matrix B, using the arithmetic rules Shimbel prescribed above. For the equations to work, the two square matrices must be of the same order.

The theorem that Shimbel proposed in this paper is: “If S is the structure matrix of a net satisfying the “general net” conditions, then the product of S with itself, namely, S2 is a matrix giving the least distances between sites, given that the path from one site to another may lead through a third” (Shimbel, 1954, 202).

Using the aforementioned minimum-sum algebra, Shimbel proves that the combined matrix S2 contains the shortest two-link path from node i to node j. Using this theorem, Shimbel clarifies that in the shortest path matrix, other nodes may have to be traversed in the process of going from point i to point j. This is provided in his proof that for any S, there exists an integer k such that Sk = Sk+1 (Shimbel, 1954, 202).

The algorithm in Shimbel’s paper, though seemingly simplistic, is quite robust. Unlike Dijkstra’s algorithm, formulated in 1956, Shimbel’s was able to account for negative edge weights and detect negative edge cycles. In Dijkstra’s approach to the shortest path problem, his algorithm will only account for direct paths of positive-length distances between two nodes. This can be ineffective in selecting the shortest path, especially when an indirect-path that has a negative-length distance can be used instead. Additionally, the iterative process of Shimbel’s algorithm also allows it to test for the presence of negative cycles. To test for the presence of negative cycles, for n nodes in the diagram, if during the final nth iteration the edge weight values (distances) can continue to be reduced, then a negative cycle exists. Any communication nets that contain negative cycles pose a threat to finding the *shortest* length path. This is because an infinite loop exists, rendering infinite potential routes to the “shortest path”.

**Expansion into the Bellman Ford algorithm**

Shimbel’s theorem is the basis for the Bellman Ford algorithm, a named coined after independent discoveries by both Bellman, in 1958, and Ford, in 1956, a few years after Shimbel’s paper. The contributions made in Bellman and Ford’s paper were exactly the same as those of Shimbel’s with the exception being that both authors provided more granular, step-by-step explanations into how the algorithm worked. In the Bellman Ford algorithm, the proof of correctness is denoted by two statements.

1. If distance[u] is not infinity, it is equal to the length of some path from s to u;
2. If there is a path from s to u with at most k edges, then distance[u] is at most the length of the shortest path from s to u with at most k edges.

A noteworthy feature of Shimbel’s algorithm is its striking similarity to the approaches introduced by later academics. In Bellman (1958, 2-6), using a functional equation approach, Bellman corroborated Shimbel’s finding by showing that,

1. A third node between two existing nodes can be more efficient.
2. Such shortest paths can be found through an iterative algorithm at n –1iterations.

These two points written by Bellman, which he developed independently, highlight the foresight and accuracy of Shimbel’s seminal paper.

Likewise, Ford (1956, 1-5) introduced the same iterative check process proposed by Shimbel in which the distance between two pairs of nodes continues to be minimalized until no further reduction is possible.

1. Ford proposes lower and upper bounds of 0 and ∞, respectively, to denote the distances between origin and destination nodes.
2. Ford’s minimal cut value mirrors Shimbel’s intermediary third node that exists between the origin and destination node. In = βij, Shimbel uses the minimum-sum to calculate the shortest length path, which is reflected in the equation that Ford later develops.
3. Ford’s also represents his maximum flow problems in a square matrix format, which results from the linking of two separate communication nets, matrix A and matrix B, together.

Bellman and Ford’s acknowledgements came in their publishing of more exhaustive, step-by-step explanations of the algorithm. The steps laid out in the Bellman Ford algorithm expanded upon the ideas proposed by Shimbel. For example, to explain finding the shortest path on a matrix, Shimbel tersely states, “Sk k ≥ 1 is a matrix giving the shortest paths from node i to node j in S, given that k – 1 other paths may be traversed in the process” (Shimbel, 1954, 202). Ford developed Shimbel’s aforementioned statement by explaining *how* to actually find the shortest path (Ford, 1956, 6-11).

1. To run the algorithm on a graph, first assign a node on the communication net as the origin node and give it a cost of 0 and assign path values of infinity to all other nodes.
2. Next, from the origin node, visit each edge and relax the path distance. Do this at most n - 1 amount of times, with n being the number of nodes present in the graph.
3. During each of the n – 1 iterations, ensure that the path to the node that the edge is pointing can continue to be shortened. There can be at most n - 1 edges in a path from the origin node to any other destination node in the graph.

**Contributions**

The modern packet switching process draws its fundamentals from the Bellman Ford algorithm. The algorithm is distributed because it involves a number of nodes (routers) within an autonomous system and consists of the following steps (Pal, 2013, 316).

1. Each node calculates the distances between itself and all other nodes within the autonomous system network and stores this information as a table. Each node has 1 row for destination d. The distance of the select origin node to itself is 0 and its distance to other nodes is ∞.
2. Each node sends its table to all neighboring nodes.
3. When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes.

Although the Bellman Ford does not run as quickly as its counterpart, the Dijkstra’s algorithm, its beauty comes in its ability to find the shortest path even in the presence of negative weights, something that the latter is unable to do.

The algorithm also had indirect influences on later papers, such as Paul Baran’s “On Distributed Communications” (1964). Whereas Shimbel’s application for the shortest path was presented in the context of neural networks, Baran used it to create his seminal paper on the operations behind modern computer network routing and packet switching. The shortest path had previously been used to understand neural networks and transportation systems. However, *how* it will contribute to the sending of electronic information remained a challenge. Since file sizes are split into smaller packets and individually routed to the destination, these packets may not be received in their original order. To solve for this, Baran created packet headers, which coupled with node addresses, helped to direct the packets’ paths. The design of the distributed computer network, applauded by Baran for its lower vulnerability, had already been utilized by Shimbel a decade earlier to portray neural network communications. Baran also borrowed from Shimbel’s study in the shortest path problem. To lower the cost of sending data between users, Baran found it more effective to have the data hop from user to user, as space becomes available, rather than waiting for a non-busy path. Each time that data is moved around, the network will change the handover by a fixed amount, to reflect the link cost of the transmission link (Baran, 1964, 23-25). Upon closer examination, Baran’s methodology used to create the routing system for packet switching does not come as a surprise. His research was conducted at the RAND Corporation, which is also the same organization that Ford was working in when he made his independent findings on what is now known as the Bellman Ford algorithm.

For example, the distance-vector routing algorithm used in modern packet switching was largely inspired by the Bellman Ford algorithm. In dx(y) = minv{c(x,v) + dv(y)}, dx(y) is the cost of the lowest-cost path, with x = origin node, y = destination node, and v = intermediary node (Kurose et. al, 2013, 371). In traveling from origin node x to intermediary node v, we subsequently take the least cost path from intermediary node v to destination node y, such that the lowest-cost paths will then be the minimum of c(x,v) + dv(y) taken over all intermediary nodes v. In the context of modern packet switching, the Bellman Ford algorithm allows the packets to know which router to hop onto next.

Computer scientists and behavioral economists also share similar interests in developing optimal networks, particularly through the minimization of distance and time. In economics, papers on network optimality primarily focus on improving the process of human communication. Mark Granovetter’s research paper, “The Strength of Weak Ties” (1973) analyzes the strength of sociometric ties based on two factors: the strength of interpersonal ties between people (distance) and how these ties relate to macro trends at the organization and community level (relative location of nodes). Using diagrams consisting of nodes and link distances, Granovetter argues that weak ties (longer distances) are more effective in the transmission of influences across longer distances and to a larger group of people. In weaker ties, an intermediary node (mutual friend or third party) serves the important role of connecting two unfamiliar people together. However, strong ties (shorter distances) are useful in fostering personal relationships or reciprocating favors. For Granovetter, the macro trends of social networking are related to the distance and context of relationships that exists between individuals.

This same concept can be applied to networks in computer science. In Simbel’s “Structure in Communication Nets”, the shortest path problem is inherently laying the groundwork to understand the cost and benefits of taking either direct or indirect paths. While a direct path is generally perceived to efficient and easier to program, it fails to account for caveats such as negative edge weights and intermediary nodes that may render an indirect route to be lowest-cost. That networks can be applied to behavioral economics demonstrates the broader applicability of Shimbel’s seminal paper on the shortest path problem. While short and long distances take on different meanings in the Granovetter and Shimbel’s research papers, two things are certain- (1) intermediary nodes play an important part in the connecting one entity to another and (2) tradeoffs exist in taking either direct or indirect paths.

**Non-classic paper critique**

Leveson and Turner’s “An Investigation of the Therac-25 Accidents” (1993) should be reevaluated for its current selection as a classic. The faults of this research paper lie in the absence of influential ideas; in addition, its structure as an overly lengthy report without much emphasis on *how* to execute potential solutions decreases its potential impact. Although the paper is exhaustive in its explanation of events that caused the Thearac-25 accident, some of the content were not entirely new. Prior research papers by Frederick P. Brooks and Ken Thompson have already communicated similar ways to reduce software errors as those presented in Leveson and Turner’s paper. Moreover, this paper merely highlights the consensus drawn from existing sources and presents it as a dispersed list of *what* needs to be done rather than *how* things should be done.

The authors suggest that the problems leading to the software errors stem from technical and procedural gaps.

In regards to technical gaps, Leveson and Turner’s discussions showed evidences of inadequate software engineering and risk assessment for the Therac-25 machine. They believed that safety does not exclusively refer to the quality of the software itself but also to the system in which it is being used (Leveson et. al, 1993, 39). However, Leveson and Turner’s sentiments suggest that software testing should be the primary prevention mechanism against software errors. The critique that I have against this sentiment is that over-reliance on testing creates a reactive approach that does not fully address the root causes of software errors. Emphasis should instead be placed on how to redesign the entire process of product creation, from ideation to the actual phases of production. That way, error assessment will not be seen as merely a benchmark but rather an integral part of the coding and designing process.

Secondly, Leveson and Turner did not explicitly mention expectations as it relates to total error isolation. In the research paper, they held an exhaustive testing approach, which required strict adherence to the enumerated software-engineering principles (Leveson et. al, 1993, 39). Some key points mentioned by the authors include:

1. Documentation should not be an after thought.
2. Software quality assurance practice and standards should be established.
3. Designs should be kept simple.
4. Integrate information access regarding errors into the software.
5. Software should be subjected to extensive testing.

Instead of focusing on the idea of being able to find and eliminate all potential errors, it may be more practical to take on a targeted testing approach. Since it is unrealistic to expect error-free software in complicated systems, a more economical approach might be to allot resources to correct errors in high-impact codes while allowing all other codes to continue to run within acceptable bounds despite the presence of error.

In regards to ethics, I find Leveson and Turner’s reference to the Therac-25 as an “accident” contradictory the very essence of their research paper, which aims to codify all aspects of software development. A careful examination of the series of events related to the Therac-25 Accident suggests that some of the errors could have been prevented by more careful human intervention. That the authors stops short of critically examining the equipment’s manufacturer and associated personnel suggests that Leveson and Turner have not fully accepted the viable role that humans play in error intervention (Leveson et. al, 1993, 18). For example, it was mentioned that patients were often the first point of contact for the machine’s errors since no independent checks were made on the software prior to the treatment sessions (Leveson et. al, 1993, 21). However, the authors did not make any attempts to elaborate on how to effectively engage human users or professionals in the software evaluation process.

For Leveson and Turner, the aforementioned topic would have been an important area of discussion. The role of humans in software development is one that was previously explored by Brooks in “The Mythical Man-Month” (1975). Brooks argues that the concept of the “man-month”, or the adding of an extra engineer into a delayed schedule, to help speed up a delayed project is misguided. For Brooks, the section of codes developed in software programming are not isolated but rather sequential tasks that is difficult to be partitioned. Since the development of codes will require human collaboration and documentation across different stages, as mentioned in Leveson and Turner’s key point (1), how will they plan to respond to potential conflicts incurred in the movement of human capital during software development? This is question was never addressed by the authors.

In regards to the second point on procedural gaps, a potential solution that was not addressed by the authors is to structure the system so that failure of subcomponents does not jeopardize other parts of the machine. This means that the system is able to survive the failure of one of its components and still operate within acceptable bounds of error. In hardware development, there are often spare components, such as processors and discs, which can serve as reliable backups. Surprisingly, Leveson and Turner make no mention of exploring backup components as it relates to software design. But in key point (4), Leveson and Turner did mention the need to integrate information access relating to code and error assessments. This proposal is not novel because it had already been mentioned two decades earlier by Brooks, who states, “we need to develop and publicize productivity figures, bug-incidence figures, estimating rules, and so on. The whole profession can only profit from sharing such data” (Brooks, 1975, 21).

Furthermore, the steps that were proposed by Leveson and Turner’s research are essentially verbatim of those steps mentioned by Brooks “No Silver Bullet” (1986).

1. Use rapid prototyping in a planned iterations to establish software requirements.
2. Grow software organically, adding and testing incrementally.
3. Identify and develop new conceptual designers.

Leveson and Turner’s key points (2) (3) and (5) match the aforementioned proposals made by Brooks.

In assessing the layout of the paper, the paper lacked strong focus and originality. This paper should instead be substituted for a paper that explains the underlying software issues but also offers well-researched solutions, or at the least, start a dialogue regarding the solution. A better-written selection might be one that integrates theorems and software development proposals from either past or current computer scientists. Finally, the disclaimer made by Leveson and Turner in the end of the paper raises concerns. The authors noted that the exact methodology used to conduct the final safety reports by the Atomic Energy of Canada Limited (AECL) were unknown (Leveson et. al, 1993, 36). They followed up with the possibility that either quantitative studies or qualitative studies (simply reading through the code) may have been used. If it is indeed the later, then it raises concerns about the accuracy of Leveson and Turner’s sources. As Thompson (1984) warned, a person cannot trust a program that he or she did not write and simply reading through a code is not enough to determine its trustworthiness. No compromise was ever mentioned by Leveson and Turner concerning the use of quantitative and qualitative error-checking mechanisms.

**Conclusion**

While the literal definition of a classic paper refers to the length of time since publication, my selection of Alfonso Shimbel’s 1954 paper titled “Structure in Communication Nets” were based on two factors, (1) its influence on the field of computer networks and (2) the clarity and foresight by which the ideas were presented in the paper. Alfonso Shimbel was a researcher in mathematical biology at the University of Chicago and used his background in neural network research to lay out the framework to solve the shortest route problem. Shimbel developed an algorithm through which it became possible to find the shortest route between any two nodes. Although he never received his due credit, Shimbel’s algorithm exactly reflected the findings of Bellman and Ford, who made their discoveries a few years. His algorithm now serves as the basis for modern packet switching. Leveson and Turner’s “An Investigation of the Therac-25 Accidents” (1993), though an interesting read, does not resonate to me as a classic. Its lack of original thought and unfocused structure makes it difficult for the reader to draw the connection between the paper’s content with those of other existing research papers. Much of Leveson and Turner’s proposals to reduce software error were already explored by Brooks and Thompson decades earlier. Its contents read more like a technical report of *what* caused the Therac-25 Accident than an actual research paper that answered *how* scientists can reduce software errors.

**Works Cited**

Baran, Paul. *On Distributed Communications I. Introduction to Distributed*

*Communication Networks*. The RAND Corporation, 1964, pp. 1–37, *On Distributed Communications I. Introduction to Distributed Communication Networks*.

Bellman, Richard. *On a Routing Problem*. The RAND Corporation, 1956, pp. 1–6, *On a*

*Routing Problem*.

Brooks, Frederick. *No Silver Bullet- Essence and Accident on Software Engineering*. pp.

1–16, *No Silver Bullet- Essence and Accident on Software Engineering*.

Brooks, Frederick. “The Mythical Man-Month.” *The Mythical Man-Month: Essays of*

*Software Engineering*, Addison-Wesley, 1975, pp. 13–26.

Ford, L.R. *Network Flow Theory*. The RAND Corporation, pp. 1–12, *Network Flow*

*Theory*.

Granovetter, Mark S. “The Strength of Weak Ties.” American Journal of Sociology, vol.

78, no. 6, May 1973, pp. 1–30.JSTOR.

Kurose, James F., and Keith W. Ross. *Computer Networking: A Top-Down Approach*.

6th ed., Pearson, 2013.

Leveson, Nancy G., and Clark S. Turner. “An Investigation of the Therac-25

Accidents.” *Computer*, vol. 26, no. 7, 7 July 1993, pp. 18–41.

Pal, Ajit. *Data Communication and Computer Networks*. PHI Learning, 2013.

Schrijver, Alexander. “On the History of Combinatorial Optimization.” *Handbooks in*

*Operations Research and Management Science*, vol. 12, 2005, pp. 1–57.

Shimbel, Alfonso. “Structure in Communication Nets.” *Proceedings of the Symposium of*

*Information Networks*, 12 Apr. 1954, pp. 199–203.

Thompson, Ken. *Reflections on Trusting Trust*. 8th ed., vol. 27, Communications of the

ACM, 1984, pp. 761–763, *Reflections on Trusting Trust*.