A Neural-network Algorithm for All *k* Shortest Paths Problem.

Kun Zhao
Department of Mathematics and Statistics
Georgia State University
Atlanta, Georgia 30303
kenzhao100@hotmail.com

Abdoul Sylla
Enterprise Architecture, Travelport Inc.
300 Galleria Pkwy SE #400
Atlanta, GA 30339

ABSTRACT

One of the fundamental computations for network analysis is to calculate the shortest path (SP) and all k shortest paths (KSP) between two nodes. Finding SP and KSP in a large graph is not trivial since the computation time increases as the number of nodes and edges increases. A recent neural network algorithm for calculating SP showed its advantage of not depending on the number of nodes and edges but the topology of a graph. Reasonable performance of the algorithm was reported. However, this algorithm is limited to the SP problem. This paper reports the progress of extending a neural network algorithm to solve the KSP problem. How to apply the new KSP algorithm to a whole-genome sequencing problem is also discussed.

Categories and Subject Descriptors

F.1.1 [**Theory of Computation**]: Models of Computation; G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations; G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problem*; J.3 [**Computer Applications**]: Biology and Genetics

General Terms

Algorithm; Design; Theory

Keywords

Graph, algorithm, differential equations, neural network, KSP problem, whole-genome sequencing, poxvirus

1. INTRODUCTION

One of the fundamental computations for network (a.k.a. graph) analysis is to calculate the shortest path (SP) between two nodes. Computing the all k shortest paths (KSP) in addition to the SP usually provides more insightful information about the graph. Finding all k shortest paths in a simple graph requires not only to determine the shortest path from the initial node to the end node, but also need to calculate the second, third, ..., kth shortest path if they are available. Many biological problems can be converted into a KSP problem after underlying graph representation is defined. For example, interaction information network [5] was constructed for analyzing biological response data via the integration of the gene expression data. By utilizing a KSP algorithm, scientists could predict particular response sub-networks in the *Mycobacterium tuberculosis* network [5]. KSP algorithms have been applied to the analysis of protein functional network [12], molecular interaction network [9], sequence alignment [6] and gene network [15]. Besides KSP's applications to biology, motivation for studying the KSP problem also comes from its intrinsic mathematical value. Computation complexity increases as the size of the problem (or graph) increases. Tremendous efforts have been made towards solving this problem [2, 6, 10, 13, 20]. Unfortunately, the computation challenges remain [14].

Interestingly, a recent neural network algorithm applicable to the SP problem showed its advantage of not depending on the number of nodes and edges but the topology of a graph [17]. Reasonable performances of the algorithm were reported. However, this algorithm is limited to the SP problem. A report regarding applying neural network algorithm for solving KSP problem was not found.

This paper reports the progress of extending a neural network algorithm to solve the KSP problem. Firstly, this study introduces a generalized neural network framework, which provides a theoretical foundation. Secondly, a neural network algorithm, namely Neural-KSP, is proposed and applied to a simple example to demonstrate how the algorithm works. Thirdly, the performance of this algorithm is analyzed. Finally, a potential application to a whole-genome sequencing problem is discussed.

2. COUPLED NEURAL NETWORK IN CONTINUOUS TIME AND KSP PROBLEM

Coupled neural networks in continuous time have been suggested as effective methods [8, 16, 3] for solving SP problem. In these methods, decision variables v_{ij} (or edges in a graph) are represented by the activation states of neurons which are further modeled by a system of differential equations. A Lyapunov (energy) function is defined to drive each neuron into its stable state [1].

Unlike modelling neurons as edges in a graph, recent techniques represent neurons as vertices in the graph [17, 18]. The dynamics of each neuron are modeled by differential equations, which are designed to realize that the smaller coupling strength (e.g. connection weights in a graph) lead to earlier firing times of an individual neuron. In other

ACMBCB '13 Washington DC, USA

http://dx.doi.org/10.1145/2506583.2506696.

words, after an excitation from the initial neuron (or node), the signal will spread (like a wave propagation) based on the graph (network) topology and individual dynamics. By constructing a proper neural network, a tracked signal travels from an initial node to terminal node through the shortest path. An advantage of the method is that the spreading time of the signal (wave) is independent of the number of nodes in the graph but only determined by the path length from the initial node to the terminal node. To find the shortest path, the individual dynamics have to be calculated. Next, we will provide a generalized form, which considers both individual dynamics and network topology, and thus offers a mathematical foundation for the Neural-KSP algorithm.

2.1 General form of coupled neural network in continuous time

To solve the KSP problem, we construct a network of n interacting linear/non-linear l-dimensional dynamical system (neurons) and denote them as $x_i = (x_i^1,\ x_i^2,\ ...,\ x_i^l)$, $i = 1, ..., n$, then we can define the following general framework for the coupled network:

$$\dot{x}_i = F(x_i) - \sum_{j=1,\ j\neq i}^{n} d_{ij}(t)\tau(x_j) \qquad (1)$$

where $F(x_i)$ define each individual system, $\tau(x_j)$ is a activation (sigmoid) function $\tau(x_j) = \frac{1}{1+e^{-\lambda(x_j-\theta)}}$. It represents that the ith neuron is excited by jth neuron while the potential of jth neuron exceed a synaptic threshold θ. $d_{ij}(t)$ is the coupling strength (weights/costs) from neuron i to neuron j, which is a positive number and could depend on time t. The logic behind this is to make sure while signal travels from neuron i to neuron j at a faster rate where the coupling strength is smaller. The advantage of providing this form is that the existing methods which model vertices in a graph as neurons (a.k.a. dynamical systems), can be included under this form.

3. A NEURAL NETWORK ALGORITHM FOR KSP PROBLEM

In general, the proposed systems can not be solved analytically and the sigmoid function complicates the problem. Solutions in a closed form was reported in a special case where the system is linear first-order differential equations (please refer [17]). An additional algorithm was also reported while the existence of the step function which plays the same role as our sigmoid function. In fact, when computer handles this problem, it needs to break the complex problem from continuous time to discrete time.

3.1 An algorithm for KSP problem under the general form

We describe the Neural-KSP algorithm, which can help to find the all k shortest paths in an acyclic graph. It can be applied to both directed and undirected graph as the graph remains loopless. Given a connected graph $G = (V, E)$, let V denote the set of neurons and E denote the set of connections between any two neurons. Again, the dynamics of neuron i is modeled by $F(x_i)$ in equation 1 and similarly d_{ij} denotes the weight between neuron i and j on a graph. A simple pseudo-code below is a discrete version of integrating equation 1.

Input: Acyclic graph with/out multiple edges, initial neuron, terminal neuron and the value of k. **Output:** all k shortest paths.
begin:
1. [initialize]
Initialize the network: excite the initial neuron. $c = 0$; $w = 0$.
2. [record the excitation status of each neuron at a specific time]
find minimal weighted outgoing edge(s) for the excited neuron(s). denote the weight as w.
record current excitation state of each neuron.
if the excited neurons are not being excited at the current time; go to 3. (Appendix example steps 1-3)
else record the excitation of the current neuron and attached all outgoing edges back to the neuron; go to 3. (Appendix example steps 4-5)
3. [run the network dynamics]
excite the downstream neuron(s).
if terminal neuron reached.
record the path(s); set $c = c + 1$;
if $c = k$; go to 4;
else all outgoing edges of the excited neurons minus w. go to 2.
4. [end]

3.2 Simple example in discrete time

We show how a neuron network can help to find the shortest path using a simple example. This is important for understanding that why the Neural-KSP algorithm is an exact algorithm. By "exact algorithm", we mean that this algorithm is able to find the global optimal solution if it exists.

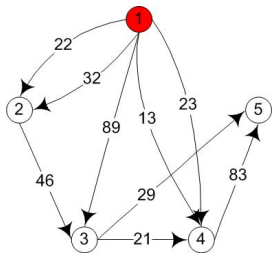


Figure 1: Example of Neural-KSP algorithm. Starting of the network: red node represents the firing neuron.

Let's consider a simple loopless directed graph with multiple edges in Fig. 1. There are nine edges with various weights. We want to find the all k shortest paths from node 1 to node 5 with respect to the minimum weights. The all k shortest paths ($k = 8$) will be (from the lowest weights to greatest weights) 1-4$_{(1)}$-5, 1-2$_{(1)}$-3$_{(1)}$-5, 1-4$_{(2)}$(2)-5, 1-2$_{(2)}$(2)-3$_{(2)}$-5, 1-3$_{(3)}$-5, 1-2$_{(1)}$-3$_{(1)}$-4$_{(3)}$-5, 1-2$_{(1)}$(2)-3$_{(2)}$-4$_{(4)}$-5, 1-3$_{(3)}$-4$_{(5)}$-5. In this example, 1-4$_{(2)}$(2)-5, represents that neuron 1 excites neuron 4 using the edge weight of 23. (2) represents the second lowest weight from node 1 to node 4. It is the second time that neuron 4 is excited, which is represented by the footnote $_{(2)}$. The last operation is shown in the Fig 2. A complete calculation of this example can be found in the APPENDIX.

4. RESULTS

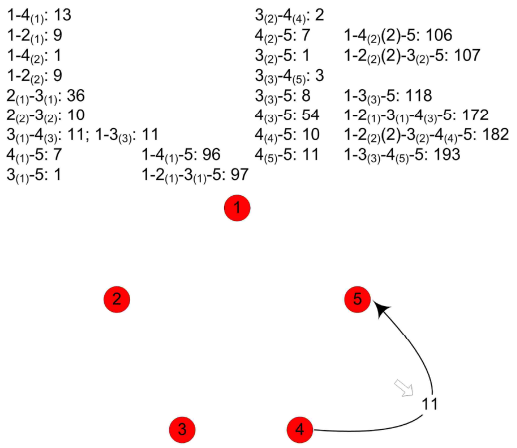


Figure 2: Example of Neural-KSP algorithm. Ending of the network: all neurons are excited, and therefore all red.

We ran experiments to evaluate the performance of the Neural-KSP algorithm for calculating all k shortest paths from the initial node to the ending node. The network data was randomly generated using MATLAB and the algorithms were implemented in C++. The execution time was collected based on an average of ten runs on each graph and using an IBM laptop running Windows 7, with 1.83Mhz Intel CPU and 4GB RAM.

Results from the Neural-KSP were compared to a Dijkstra-based KSP algorithm, namely "Near-KSP"[4, 19]. Dijkstra's algorithm finds the shortest path from an initial node to all other nodes. The algorithm works by keeping, for each node N, Cost[N] is the cost of the shortest path found so far between the initial node and the node N. At the beginning, this value is 0 for the initial node (i.e., Cost[initial]=0), and non-existant (i.e., infinity) for all other nodes. When the algorithm finishes, Cost[N] will be the cost of the shortest path from the initial to N or non-existant, if no such path exists. Link just keep track of the path from the initial node to the other nodes [4]. The Near-KSP algorithm, as an extension of Dijkstra algorithm for solving KSP problem, was reported with reasonable performances and easy to implement [4, 19].

As shown in the Table 1, in a special case where $k = 1$, the Neural-KSP perform better than the Near-KSP algorithm in a larger network and the path length is short (only two or three) on four randomly generated networks. These observation is consistent with previous reports [17]. The new ingredient in Table 1 is that the Neural-KSP algorithm could be efficient to the KSP problem (while $k = 5, 10, 15$) where the graphs have multiple edges, short paths and larger scales. To understanding the algorithm further, we started with the network of 1000 nodes and 1,996,825 multiple edges in the Table 1, by deleting/adding a few edges from the initial network (in a way that we can increase the minimal path length), we obtained four additional networks in Table 2. The results in this table confirmed that Neural-KSP performs better when network has short path lengths, as Near-KSP algorithm performs better if one knows that all k shortest paths is not short. (For example, the networks have a path length great than four in Table 2.) We are optimistic about these results because we believe that the performance of each algorithm could be further improved by the hybridizing the two.

Table 1: Performance comparison on four multi-graphs with different node and edge numbers.

K[1]	Nodes	Edges	Near[2]	Neural[3]	PL[4]
1	5	14	0	0.0015	2
	100	149	0	0.0015	3
	500	500084	35	3	2
	1000	1996820	137	11	2
5	5	14	0.0015	0.0016	3
	100	149	0.015	0.016	4
	500	500084	32	9	2
	1000	1996820	138	24	2, 3
10	5	7	0.004	0.005	2, 3, 4
	100	149	0.015	0.031	4, 5
	500	500084	34	15	2
	1000	1996820	139	41	2, 3
15	5	7	0.015	0.016	3
	100	149	0.03	0.04	4
	500	500084	34	26	2
	1000	1996820	140	42	2,3

[1] All k shortest paths; [2]Near KSP algorithm in terms of CPU time in second; [3] The Proposed Neural-KSP algorithm in terms of CPU time in second; [4]Path Length.

Table 2: Performance comparison on five multi-graphs with same node number.

K [1]	Nodes	Edges	Near [2]	Neural [3]	Min PL[4]
1	1000	1996825	132	1	1
		1996820	137	11	2
		1996812	138	13	3
		1996804	141	67	4
		1996800	150	101	5
5	1000	1996825	142	14	2
		1996820	138	24	3
		1996812	146	27	3
		1996804	142	216	4
		1996800	143	278	5

[1] All k shortest paths; [2]Near KSP algorithm in terms of CPU time in second; [3] The Proposed Neural KSP algorithm in terms of CPU time in second; [4]Minimal path length in all k shortest paths.

In short, the previous reported neural network algorithm have been generalized in a mathematical neural network framework. The neural network algorithm has been extended to solve the KSP problem where graphs have multiple edges. The computation time of Neural-KSP algorithm depends on the network topology. It performs better while the path length of all k shortest paths is small and the network scale is large. Neural-KSP algorithm is a potential solution to the network where one has the prior knowledge of the existence of short path lengths in a network. Next, we will illustrate applications to whole-genome sequencing (WGS).

5. APPLICATION TO WHOLE-GENOME SEQUENCING

A new landscape for combatting infectious diseases is to

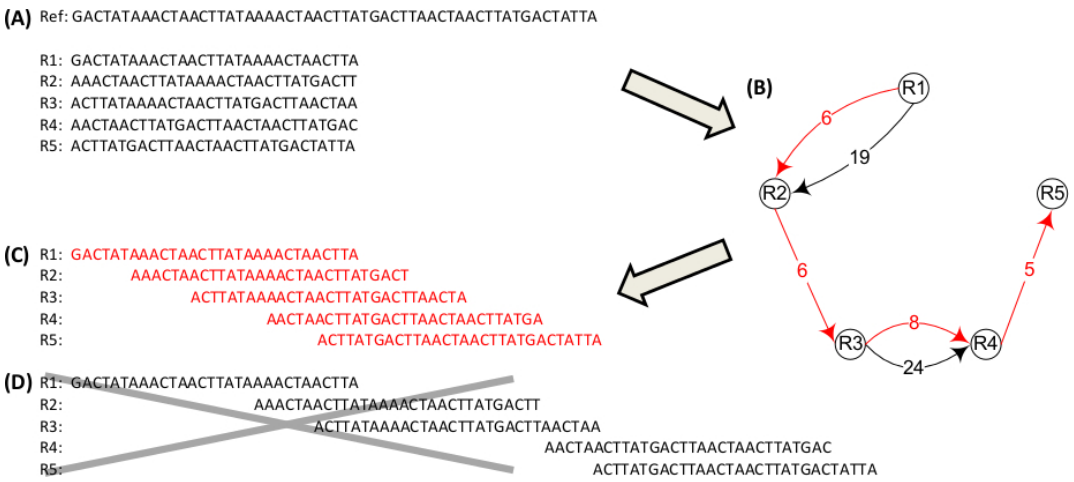


Figure 3: DNA sequence can be predicted correctly by the using shortest path. (A) Ref is a reference DNA sequence in the monkeypox genome (accession number: DQ011154), positioned from 4809 to 4863. Sequence R1 to sequence R5 mimic short fragments observed from experiments. (B) An overlapping graph. Nodes represent sequences (R1 to R5). If there is an overlap between two sequences, then an edge is added. Weight is assigned according to the non-overlapping nucleotides between two overlapping sequences. Red path is the correct assembly, as shown in (C). Black path is an incorrect assembly as shown in (D).

advance in science and technology aimed at identifying the complete genetic makeup of microorganisms. Recent reports highlighted early efforts of the use of the powerful whole-genome sequencing (WGS) technology to diagnose infectious diseases and identify outbreak pathogens with increased timeliness and accuracy at decreased costs [7].

The emerging WGS technology provides a great opportunity to apply the Neural-KSP algorithm to predict the genome sequence. The WGS technology generates millions of shorter sequence fragments. These fragments could be merged and ordered into a longer sequence, namely "contig"(or sometimes called assembly). The first step for obtaining the whole genome sequence from millions of short fragments is to construct highly ranked, high-quality contigs. The problem of constructing and ranking several contigs in between two particular short fragments from millions of fragments with similar lengths, is equivalent to finding all k shortest paths in a graph. One of the classical approaches to this problem is the overlapping graph method. According to the method, a node in a graph represents a short fragment. If there is an overlap between two fragments then an edge is added. A weight is also assigned by taking into consideration the overlapping length and other factors. A contig is a path in the graph between two nodes. The highest ranked contig corresponds to the shortest path between two sequence fragments, and the second highest corresponds the second shortest path and so on. Therefore, by referencing the information from the all k shortest paths in the overlapping graph, scientists could obtain first-hand information regarding a whole-genome sequence.

Here we use two DNA sequences from the monkeypox whole-genome sequence deposited in the National Center for Biotechnology Information (NCBI) as examples. Poxviruses are the largest and most complex known virus and infect humans with four genera: orthopox, parapox, yatapox, molluscipox. Monkeypox is a member of the orthopox and was first recognized outside Africa in 2003 during an outbreak in the United States that was traced to imported monkeypox virus (MPXV)-infected West African rodents [11]. In Africa, monkeypox has killed between 1% and 10% of people who contract it. Although people in Africa who received the smallpox vaccine have a lower risk of getting monkeypox, there is no specific treatment for monkeypox. The utilization of WGS technique as a tool for clinical molecular diagnostics offers a quick and accurate way to identify monkeypox. Unfortunately, the computational challenge lies in assembling the repeats regions to the existing *de novo* assembling algorithms and the hairpin loops (a unique characteristics of poxvirus) of many published genome are still missing. Next, we will show how to convert a whole-genome sequencing problem to KSP problem in the simplest way, and then we can apply the Neural-KSP algorithm.

In Figure 3(A), the reference DNA sequence (Ref in the figure) have 55 nucleotides from the monkeypox (accession number: DQ011154) from position 4809 to 4863. The sequences R1 to R5 mimic the experimental data from a short-read sequencing platform (for example, Illumina). Each sequence has 30 nucleotides, which are obtained from the reference sequence. For simplicity, let's assume the following: 1. The only data we have are R1 to R5; 2. The sequenced nucleotides are reliable (no errors, mutations, insertions or deletions); 3. We know the genome sequence starts from R1 and ends at R5 and we want to use as many of these short fragments as possible. The goal is to get the correct sequence assembly information. We build a graph 3(B) based on the non-overlapping lengths. For example, the edge from R1 to R2 with a weight of 6 in Figure 3(B), corresponds to the 6 non-overlapping nucleotides in the alignment of R1 and R2 in Figure 3(C). In this example, since the alignment is not unique, we can also align these fragments into 3(D). In Figure 3, the path in red corresponds to the shortest path utilizing all fragments. In this case, by calculating shortest path, we can obtain the correct assembly. However, after we calculated the shortest path sequence in Figure 4(B), we found that the shortest path in Figure 4(D) did not tell us the correct sequence assembly. It is necessary to calculate all k shortest paths in this example. Thus, by using the Neural-KSP algorithm and performing the calcu-

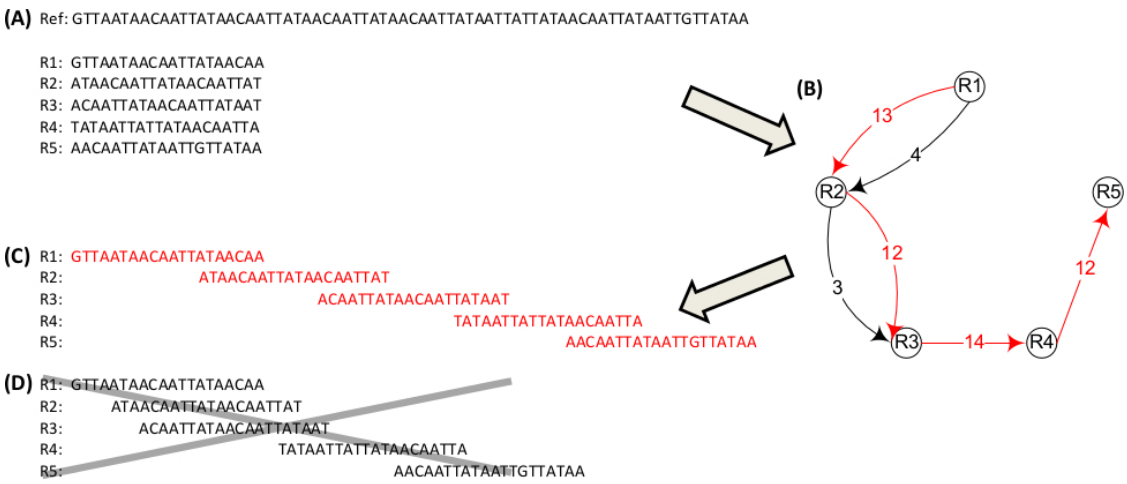


Figure 4: DNA sequence cannot be predicted correctly by using the shortest path. (A) Ref is a reference DNA sequence in the monkeypox genome (accession number: DQ011154), positioned from 142075 to 142145. Sequence R1 to sequence R5 mimic short fragments observed from experiments. (B) An overlapping graph. Nodes represent sequences (R1 to R5). If there is an overlap between two sequences, then an edge is added. Weight is assigned according to the non-overlapping nucleotides between two overlapping sequences. Red path is the correct assembly, as shown in (C). Black path is an incorrect assembly as shown in (D).

lation similar to the example in the APPENDIX, we can obtain the correct DNA sequence.

6. REFERENCES

[1] C. Ahn, R. Ramakrishna, C. Kang, and I. Choi. *Electronics Letters*, 37(19):1176–1178, 2001.

[2] K. N. Androutsopoulos and K. G. Zografos. Solving the k-shortest path problem with time windows in a time varying network. *Operations Research Letters*, 36(6):692 – 695, 2008.

[3] F. Araujo, B. Ribeiro, and L. Rodrigues. *IEEE Transactions on Neural Networks*, 12(5):1067–1073, 2001.

[4] T. Budd. *Data Structures in C++*. Addison-Wesley., 1994.

[5] L. Cabusora, E. Sutton, A. Fulmer, and C. Forst. Differential network expression during drug and stress response. *Bioinformatics*, 21(12):2661–2672, 2005.

[6] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652ÍC673, 1998.

[7] Y. Grad, M. Lipsitch, M. Feldgarden, et al. Genomic epidemiology of the escherichia coli o104:h4 outbreaks in europe, 2011. *Proc Natl Acad Sci USA*, 109(14):5547, 2012.

[8] J. Hopfield and D. Tank. *Biol Cybern*, 52(3):141–152, 1985.

[9] T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18(Suppl. 1):S233ŰS240, 2002.

[10] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12(4):411ÍC427, 1982.

[11] A. Likos, S. Sammons, V. Olson, et al. A tale of two clades: monkeypox viruses. *J Gen Virol*, 86(10):2661–2672, 2005.

[12] K. G. Mawuenyega, C. V. Forst, K. M. Dobos, et al. Mycobacterium tuberculosis functional network analysis by global subcellular protein profiling. *Mol. Biol. Cell*, 16:396–404, 2005.

[13] M. Pollack. The kth best route through a network. *Operations Research*, 9(4):578–580, 1961.

[14] A. Sedeno-Noda and C. Gonzalez-Martin. On the k shortest path trees problem. *SIAM Journal on Computing*, 202(3):628 – 635, 2010.

[15] Y. Shih and S. Parthasarathy. A single source k-shortest paths algorithm to infer regulatory pathways in a gene network. *Bioinformatics*, 28(12):i49–i58, 2012.

[16] C. Wang, T. Chen, N. Zhang, et al. Melittin, a major component of bee venom, sensitizes human hepatocellular carcinoma cells to tumor necrosis factor-related apoptosis-inducing ligand (trail)-induced apoptosis by activating camkii-tak1-jnk/p38 and inhibiting ikappabalpha kinase-nfkappab. *J Biol Chem*, 284(6):3804–13, 2009.

[17] X. Wang, H. Qu, and Y. Zhang. A modified pulse coupled neural network for shortest-path problem. *Neurocomputing*, 72(13):3028–3033, 2009.

[18] Y. Wang, G. Wu, L.and Wei, and S. Wang. *Digital Signal Processing*, 21:517–521, 2001.

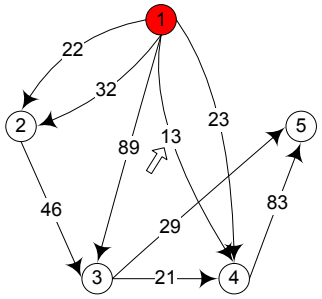
[19] M. Waterman and T. Byers. A dynamic programming algorithm to find all solutions in a neighborhood of the optimum. *Mathematical Biosciences*, 77:179–188, 1985.

[20] J. Yen. Finding the shortest loopless paths in a network. *Managment Science*, 17(11):712–716, 1971.

APPENDIX

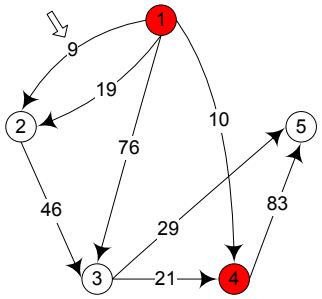
Step 1

1-4$_{(1)}$: 13



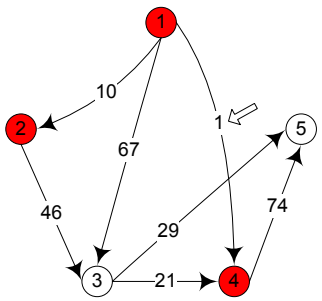
Step 2

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9



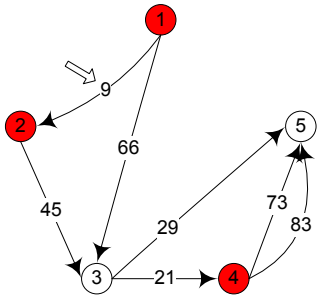
Step 3

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1



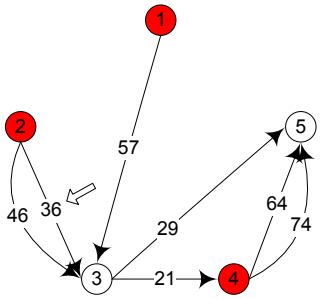
Step 4

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9



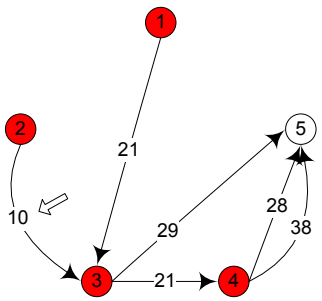
Step 5

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36



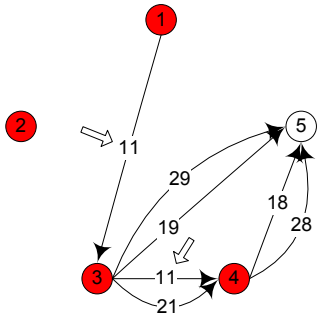
Step 6

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36
2$_{(2)}$-3$_{(2)}$: 10



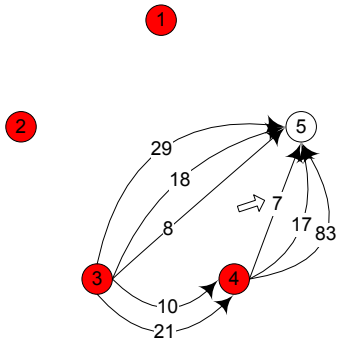
Step 7

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36
2$_{(2)}$-3$_{(2)}$: 10
3$_{(1)}$-4$_{(3)}$: 11; 1-3$_{(3)}$: 11



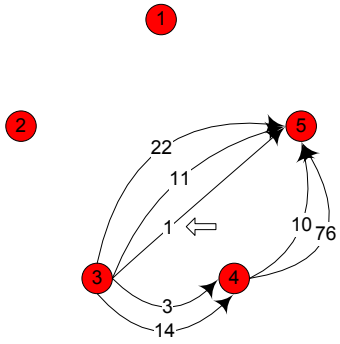
Step 8

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36
2$_{(2)}$-3$_{(2)}$: 10
3$_{(1)}$-4$_{(3)}$: 11; 1-3$_{(3)}$: 11
4$_{(1)}$-5: 7 1-4$_{(1)}$-5: 96



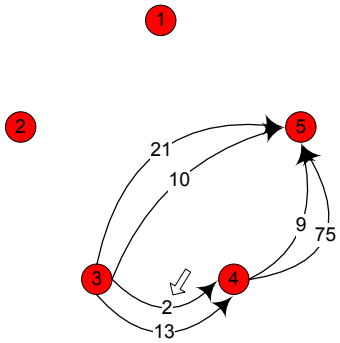
Step 9

1-4$_{(1)}$: 13
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36
2$_{(2)}$-3$_{(2)}$: 10
3$_{(1)}$-4$_{(3)}$: 11; 1-3$_{(3)}$: 11
4$_{(1)}$-5: 7 1-4$_{(1)}$-5: 96
3$_{(1)}$-5: 1 1-2$_{(1)}$-3$_{(1)}$-5: 97



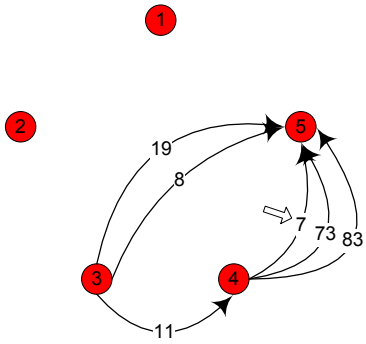
Step 10

1-4$_{(1)}$: 13 3$_{(2)}$-4$_{(4)}$: 2
1-2$_{(1)}$: 9
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36
2$_{(2)}$-3$_{(2)}$: 10
3$_{(1)}$-4$_{(3)}$: 11; 1-3$_{(3)}$: 11
4$_{(1)}$-5: 7 1-4$_{(1)}$-5: 96
3$_{(1)}$-5: 1 1-2$_{(1)}$-3$_{(1)}$-5: 97



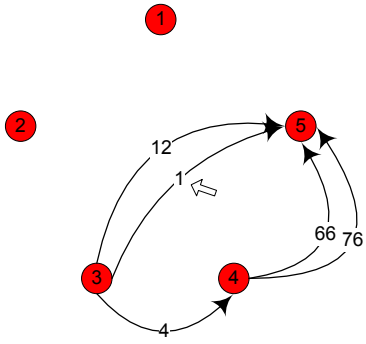
Step 11

1-4$_{(1)}$: 13 3$_{(2)}$-4$_{(4)}$: 2
1-2$_{(1)}$: 9 4$_{(2)}$-5: 7 1-4$_{(2)}$(2)-5: 106
1-4$_{(2)}$: 1
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36
2$_{(2)}$-3$_{(2)}$: 10
3$_{(1)}$-4$_{(3)}$: 11; 1-3$_{(3)}$: 11
4$_{(1)}$-5: 7 1-4$_{(1)}$-5: 96
3$_{(1)}$-5: 1 1-2$_{(1)}$-3$_{(1)}$-5: 97



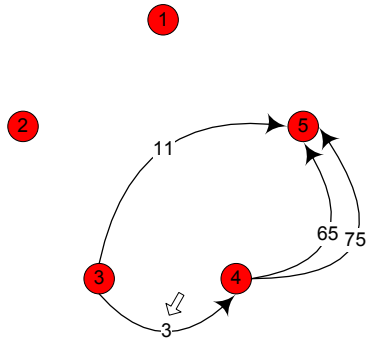
Step 12

1-4$_{(1)}$: 13 3$_{(2)}$-4$_{(4)}$: 2
1-2$_{(1)}$: 9 4$_{(2)}$-5: 7 1-4$_{(2)}$(2)-5: 106
1-4$_{(2)}$: 1 3$_{(2)}$-5: 1 1-2$_{(2)}$(2)-3$_{(2)}$-5: 107
1-2$_{(2)}$: 9
2$_{(1)}$-3$_{(1)}$: 36
2$_{(2)}$-3$_{(2)}$: 10
3$_{(1)}$-4$_{(3)}$: 11; 1-3$_{(3)}$: 11
4$_{(1)}$-5: 7 1-4$_{(1)}$-5: 96
3$_{(1)}$-5: 1 1-2$_{(1)}$-3$_{(1)}$-5: 97



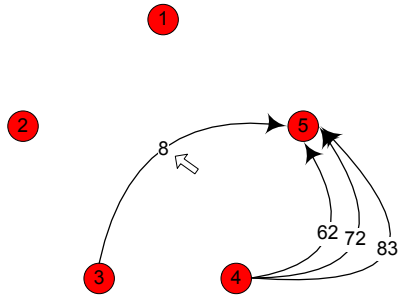
Step 13

$1\text{-}4_{(1)}$: 13
$1\text{-}2_{(1)}$: 9
$1\text{-}4_{(2)}$: 1
$1\text{-}2_{(2)}$: 9
$2_{(1)}\text{-}3_{(1)}$: 36
$2_{(2)}\text{-}3_{(2)}$: 10
$3_{(1)}\text{-}4_{(3)}$: 11; $1\text{-}3_{(3)}$: 11
$4_{(1)}\text{-}5$: 7
$3_{(1)}\text{-}5$: 1
$1\text{-}4_{(1)}\text{-}5$: 96
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}5$: 97
$3_{(2)}\text{-}4_{(4)}$: 2
$4_{(2)}\text{-}5$: 7
$3_{(2)}\text{-}5$: 1
$3_{(3)}\text{-}4_{(5)}$: 3
$1\text{-}4_{(2)}(2)\text{-}5$: 106
$1\text{-}2_{(2)}(2)\text{-}3_{(2)}\text{-}5$: 107



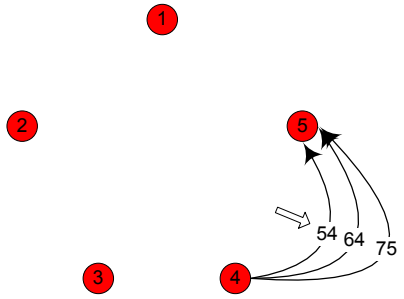
Step 14

$1\text{-}4_{(1)}$: 13
$1\text{-}2_{(1)}$: 9
$1\text{-}4_{(2)}$: 1
$1\text{-}2_{(2)}$: 9
$2_{(1)}\text{-}3_{(1)}$: 36
$2_{(2)}\text{-}3_{(2)}$: 10
$3_{(1)}\text{-}4_{(3)}$: 11; $1\text{-}3_{(3)}$: 11
$4_{(1)}\text{-}5$: 7
$3_{(1)}\text{-}5$: 1
$1\text{-}4_{(1)}\text{-}5$: 96
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}5$: 97
$3_{(2)}\text{-}4_{(4)}$: 2
$4_{(2)}\text{-}5$: 7
$3_{(2)}\text{-}5$: 1
$3_{(3)}\text{-}4_{(5)}$: 3
$3_{(3)}\text{-}5$: 8
$1\text{-}4_{(2)}(2)\text{-}5$: 106
$1\text{-}2_{(2)}(2)\text{-}3_{(2)}\text{-}5$: 107
$1\text{-}3_{(3)}\text{-}5$: 118



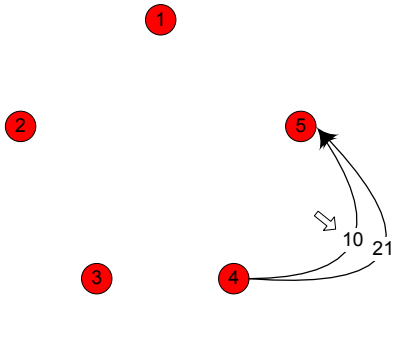
Step 15

$1\text{-}4_{(1)}$: 13
$1\text{-}2_{(1)}$: 9
$1\text{-}4_{(2)}$: 1
$1\text{-}2_{(2)}$: 9
$2_{(1)}\text{-}3_{(1)}$: 36
$2_{(2)}\text{-}3_{(2)}$: 10
$3_{(1)}\text{-}4_{(3)}$: 11; $1\text{-}3_{(3)}$: 11
$4_{(1)}\text{-}5$: 7
$3_{(1)}\text{-}5$: 1
$1\text{-}4_{(1)}\text{-}5$: 96
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}5$: 97
$3_{(2)}\text{-}4_{(4)}$: 2
$4_{(2)}\text{-}5$: 7
$3_{(2)}\text{-}5$: 1
$3_{(3)}\text{-}4_{(5)}$: 3
$3_{(3)}\text{-}5$: 8
$4_{(3)}\text{-}5$: 54
$1\text{-}4_{(2)}(2)\text{-}5$: 106
$1\text{-}2_{(2)}(2)\text{-}3_{(2)}\text{-}5$: 107
$1\text{-}3_{(3)}\text{-}5$: 118
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}4_{(3)}\text{-}5$: 172



Step 16

$1\text{-}4_{(1)}$: 13
$1\text{-}2_{(1)}$: 9
$1\text{-}4_{(2)}$: 1
$1\text{-}2_{(2)}$: 9
$2_{(1)}\text{-}3_{(1)}$: 36
$2_{(2)}\text{-}3_{(2)}$: 10
$3_{(1)}\text{-}4_{(3)}$: 11; $1\text{-}3_{(3)}$: 11
$4_{(1)}\text{-}5$: 7
$3_{(1)}\text{-}5$: 1
$1\text{-}4_{(1)}\text{-}5$: 96
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}5$: 97
$3_{(2)}\text{-}4_{(4)}$: 2
$4_{(2)}\text{-}5$: 7
$3_{(2)}\text{-}5$: 1
$3_{(3)}\text{-}4_{(5)}$: 3
$3_{(3)}\text{-}5$: 8
$4_{(3)}\text{-}5$: 54
$4_{(4)}\text{-}5$: 10
$1\text{-}4_{(2)}(2)\text{-}5$: 106
$1\text{-}2_{(2)}(2)\text{-}3_{(2)}\text{-}5$: 107
$1\text{-}3_{(3)}\text{-}5$: 118
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}4_{(3)}\text{-}5$: 172
$1\text{-}2_{(2)}\text{-}3_{(2)}\text{-}4_{(4)}\text{-}5$: 182



Step 17

$1\text{-}4_{(1)}$: 13
$1\text{-}2_{(1)}$: 9
$1\text{-}4_{(2)}$: 1
$1\text{-}2_{(2)}$: 9
$2_{(1)}\text{-}3_{(1)}$: 36
$2_{(2)}\text{-}3_{(2)}$: 10
$3_{(1)}\text{-}4_{(3)}$: 11; $1\text{-}3_{(3)}$: 11
$4_{(1)}\text{-}5$: 7
$3_{(1)}\text{-}5$: 1
$1\text{-}4_{(1)}\text{-}5$: 96
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}5$: 97
$3_{(2)}\text{-}4_{(4)}$: 2
$4_{(2)}\text{-}5$: 7
$3_{(2)}\text{-}5$: 1
$3_{(3)}\text{-}4_{(5)}$: 3
$3_{(3)}\text{-}5$: 8
$4_{(3)}\text{-}5$: 54
$4_{(4)}\text{-}5$: 10
$4_{(5)}\text{-}5$: 11
$1\text{-}4_{(2)}(2)\text{-}5$: 106
$1\text{-}2_{(2)}(2)\text{-}3_{(2)}\text{-}5$: 107
$1\text{-}3_{(3)}\text{-}5$: 118
$1\text{-}2_{(1)}\text{-}3_{(1)}\text{-}4_{(3)}\text{-}5$: 172
$1\text{-}2_{(2)}(2)\text{-}3_{(2)}\text{-}4_{(4)}\text{-}5$: 182
$1\text{-}3_{(3)}\text{-}4_{(5)}\text{-}5$: 193