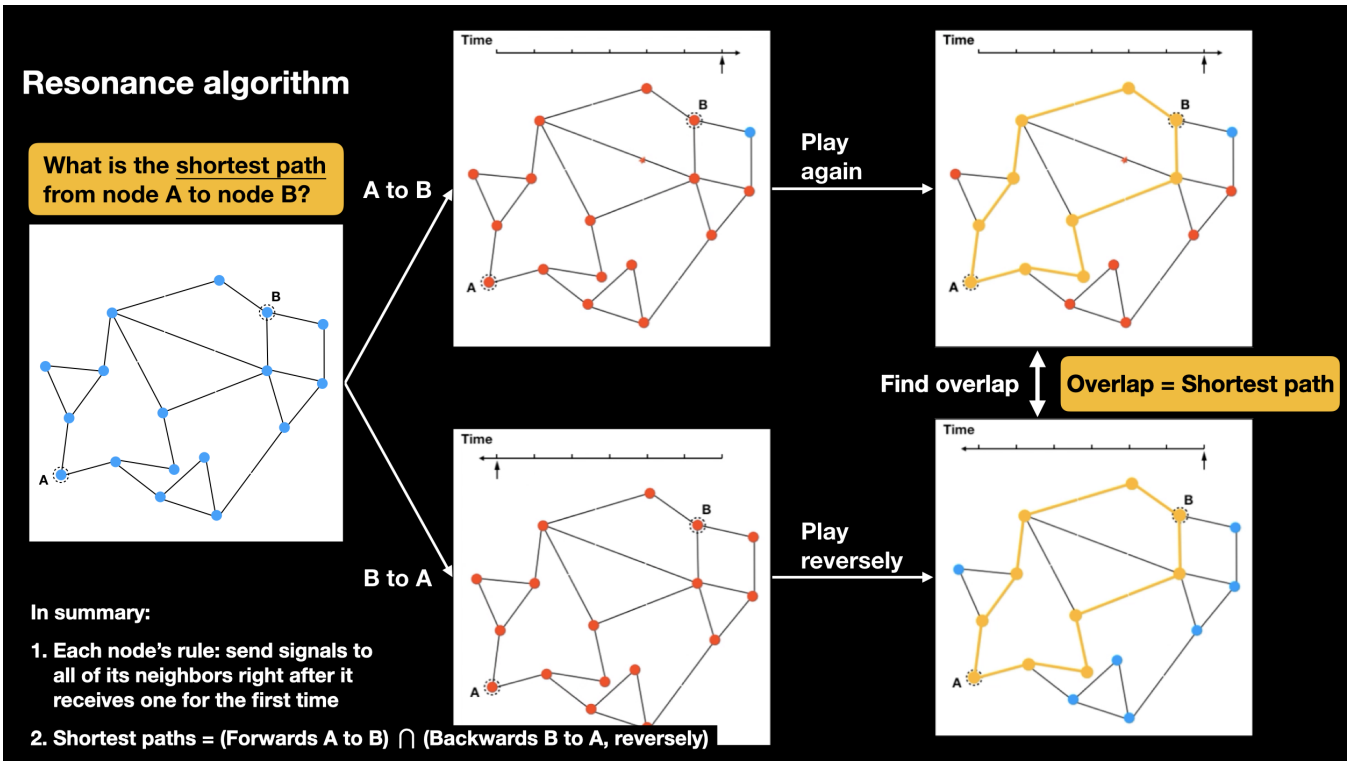
A new way to look at the shortest path problem: “Resonance algorithm”

Yu (Ernest) Liu

yu.ernest.liu@hotmail.com | https://www.wuyichen.org/resonance-algorithm | 2020.04.08



What is the shortest path from node A to node B? This is a classic shortest path problem, with various classic algorithms available [see Wikipedia]. Here I present a new algorithm—the “resonance algorithm”—with an animation and a way to implement by matrix.

The animation is available on my personal website[1], YouTube[2] and Bilibili[3], explained in details in section 1. The implementation is elaborated in section 2. In the last section, I compare this algorithm with the classic Dijkstra's algorithm, followed by some discussions.

[1] Animation on my personal website, https://www.wuyichen.org/resonance-algorithm

[2] Animation on YouTube, https://www.youtube.com/watch?v=ERNqfa_7xKY

[3] Animation on Bilibili (in Chinese), https://www.bilibili.com/video/BV1Wi4y187Cm

1. Resonance algorithm: Animation explained

To illustrate the resonance algorithm, take the map shown in Fig. 1a as an example, also referring to the animation[1,2,3]. Although Fig. 1a is a weighted undirected map (or graph), the resonance algorithm can deal with any undirected, directed or mixed graphs with unweighted or positively weighted edges.

As summarized in the animation, the following two points are key to the resonance algorithm:

- Each node's rule: Send signals to all of its neighbors right after it receives one for the first time;
- Shortest paths = (Forwards A to B) ∩ (Backwards B to A, reversely).

Now, the following three subsections elaborate the animation step by step.

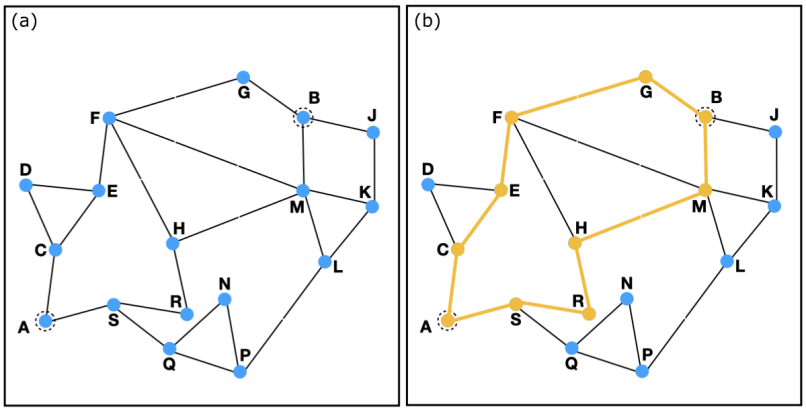


Fig. 1. (a) The exemplified map to illustrate the resonance algorithm. The question is to find the shortest path from node A to node B? This map is undirected, i.e., moving from either end of an edge to the other end is possible. Its edges are weighted, represented by lengths of the edges, which can be interpreted as the time needed to move from one end to the other. For example, it takes 1 second to move from A to C; 1 second from R to H; 2 seconds from M to H and 3 seconds from F to M. Here the weights are assumed to be integers but they could be any positive numbers. (b) The highlighted yellow paths are the two and only two shortest paths from node A to B.

1.1. Forwards A to B.

- To begin with ($t = 0$), the starting point node A sends signals to all of its neighbors simultaneously, namely nodes C and S.
- At $t = 1$, nodes C and S receive the signal. Immediately, C sends signals to its neighbors D and E; while S sends signals to its neighbors R and Q. Note that they do not send signals to the node where the signal came from.
- At $t = 2$, D, E, R and Q receive the signal. Immediately, D sends a signal to E; E sends signals to D and F; R sends a signal to H; Q sends signals to N and P.
- At $t = 3$, E, D, F, H, N and P receive the signal. Note that, although D and E receive signals again, they will not send signals because this is not the first time they receive them (which means D and E will never send signals again). Nodes F, H, N, and P send signals to their neighbors, immediately.
- This process continues, until the destination node B receives a signal. Then the forward process stops, and we obtain a video clip $\mathbf{X}$.

1.2. Backwards B to A. In the backward process, the signal is sent from node B (the original destination), until node A (the original starting point) receives a signal. All of the nodes obey the exact rules as in the forward process. Then we obtain a video clip $\mathbf{Y}$.

1.3. Find overlap. This is the last step of the resonance algorithm, after which all of the shortest paths will reveal themselves (could be only one path or many).

- In this step, we simultaneously play the video clips $\mathbf{X}$ (that records the forward process) and $\mathbf{Y}$ (that records the backward process) we have already obtained. But note that the key is to play $\mathbf{X}$ normally but play $\mathbf{Y}$ reversely.
- At each frame during playing, we mark the signals that appear at the same position in both clips.
- In the end, the paths covered by all of these marked signals are the shortest paths (referring to the highlighted yellow paths in the animation at the end, also shown in Fig. 1b).

2. Resonance algorithm: Implementing by matrix

In this section, I explain how to implement the resonance algorithm by matrix, which is much more systematic and convenient, although less intuitive than the animation. First, I summarize the key points in general:

- Forwards A to B:
    1. Initialize a zero matrix (denoted as $\mathbf{X}$ and each entry is denoted as $X_{i,j}$) where one row represents one node and one column represents one time point.
    $\mathbf{X}$ records the time point at which any node sends signals, where “1” stands for signals being sent and “0” otherwise.

2. Now $t = 0$. Set $X_{A,0} = 1$, meaning that the starting point node A sends signals at $t = 0$ to all of its neighbors; For each neighbor (denoted V), there is a specific time point (denoted T_V) it receives the signal. Set each $X_{V,T_V} = 2$ (any arbitrary number works, and this value "2" will be changed later. Here it is just for marking).
3. Go to the next time point $(t + 1)$, and do the following procedures:
   - Check the whole column $(t + 1)$ to see which entries are 2, and set them to 1, meaning that the corresponding nodes send signals to all of their neighbors at $(t + 1)$.
   - For each neighbor (denoted V) which the signals are sent to, there is a specific time point (denoted T_V) it receives the signal. Mark each $X_{V,T_V} = 2$.
     But when marking, make sure that (1) if the neighbor V has already sent signals before, do not mark it, and (2) if V will receive signals at several time points after $(t + 1)$, only mark the first time point and change others to 0.
4. Repeat from step 3 until the destination node B receives signals. Then, we obtain the matrix $\mathbf{X}$ that we want.

- Backwards B to A:
  In this backward process, the signal is sent from node B (the original destination), until node A (the original starting point) receives signals. All other rules are exactly the same as in the forward process. In the end, we obtain a matrix $\mathbf{Y}$ for this backward process.

- Find overlap:

1. We will reverse the time for the backward process:
   Create a new matrix $\mathbf{Y}'$ such that the first column of $\mathbf{Y}'$ is the last column of $\mathbf{Y}$, the second column of $\mathbf{Y}'$ is the second to last column of $\mathbf{Y}$, and so on. That is, $col_j(\mathbf{Y}') = col_{(W-j+1)}(\mathbf{Y})$ where W is the number of columns of $\mathbf{Y}$.
2. Now, we will find the overlaps between $\mathbf{X}$ and $\mathbf{Y}'$:
   Create a matrix $\mathbf{Z}$ with the same size as $\mathbf{X}$. Set each entry of $\mathbf{Z}$ to be equal to the logical conjunction of the corresponding entries of $\mathbf{X}$ and $\mathbf{Y}'$. That is, set $Z_{i,j} = X_{i,j} \wedge Y'_{i,j}$. Note that $\wedge$ represent the logical conjunction operator, i.e., $0 \wedge 0 = 1 \wedge 1 = 1$ while $0 \wedge 1 = 1 \wedge 0 = 0$.
   The matrix $\mathbf{Z}$ is all we want, that contains the information of the shortest paths. The whole algorithm is finished.

Then, by taking the exemplified map (Fig. 1a), I will elaborate step by step in the following three subsections.

2.1. Forwards A to B.

- At $t = 0$, we set $X_{A,0} = 1$, meaning that the starting point node A sends signals. We know that its neighbors C and S will receive this signal at $t = 1$, so we temporarily set $X_{C,1} = X_{S,1} = 2$ to mark the future events, as shown in Fig. 2a.

- At $t = 1$, in the whole column $t = 1$ we see $X_{C,1} = X_{S,1} = 2$ which means that only C and S receive signals at $t = 1$ (Fig. 2a). They will send signals to their neighbors immediately. We thus officially set $X_{C,1} = X_{S,1} = 1$ to denote signals being sent.
  Node C has neighbors D, E and A, so they will receive signals at $t = 2$. But we will only mark the nodes who receive signals for the first time, so we mark $X_{D,2} = X_{E,2} = 2$ (evidently, the node where the signal came from should never be marked again, which is also why in the animation, no signal will be sent back to where it came from). Similarly, we mark mark $X_{Q,2} = X_{R,2} = 2$, as shown in Fig. 2b.

- At $t = 2$, we see that D, E, R and Q receive the signal (Fig. 2b). So, they will send signals to their neighbors immediately, and we thus set $X_{D,2} = X_{E,2} = X_{R,2} = X_{Q,2} = 1$. Likewise, we mark their corresponding neighbors at the time point they receive signals: $X_{F,3} = X_{H,3} = X_{N,3} = X_{P,3} = 2$, as shown in Fig. 2c.

- At $t = 3$, F, H, N and P receive the signal (Fig. 2c), so we set $X_{F,3} = X_{H,3} = X_{N,3} = X_{P,3} = 1$. Likewise, we mark their corresponding neighbors: $X_{G,5} = X_{L,5} = X_{M,5} = 2$, as shown in Fig. 2d (notice the weighted edges, i.e., it takes 2 seconds from F to H, from H to F, from F to G, from H to M and from P to L, and takes 3 seconds from F to M).
  Note that the signal that F sent arrives at M at $t = 6$ while the signal that H sent arrives at M at $t = 5$, but we will only mark the one that arrives first, i.e., only set $X_{M,5} = 1$.

- At $t = 4$, we see no node receives signals, so we directly move to the next time step.

- At $t = 5$, G, L and M receive signals for the first time (Fig. 2d), so we set $X_{G,5} = X_{L,5} = X_{M,5} = 1$. Likewise, we mark $X_{B,6} = X_{K,6} = 2$, as shown in Fig. 2e.

- At $t = 6$, the destination node B receives signals, so the process stops. The final step is to change the non-zero values $X_{B,6}$ and $X_{K,6}$ to 1, as shown in Fig. 2f.
  The matrix $\mathbf{X}$ shown in Fig. 2f is what we need.

2.2. Backwards B to A. This is the same process as the forward one, except the signal is sent from node B (the original destination), until node A (the original starting point) receives signals. In the end, we obtain a matrix $\mathbf{Y}$, as shown in Fig. 2g.

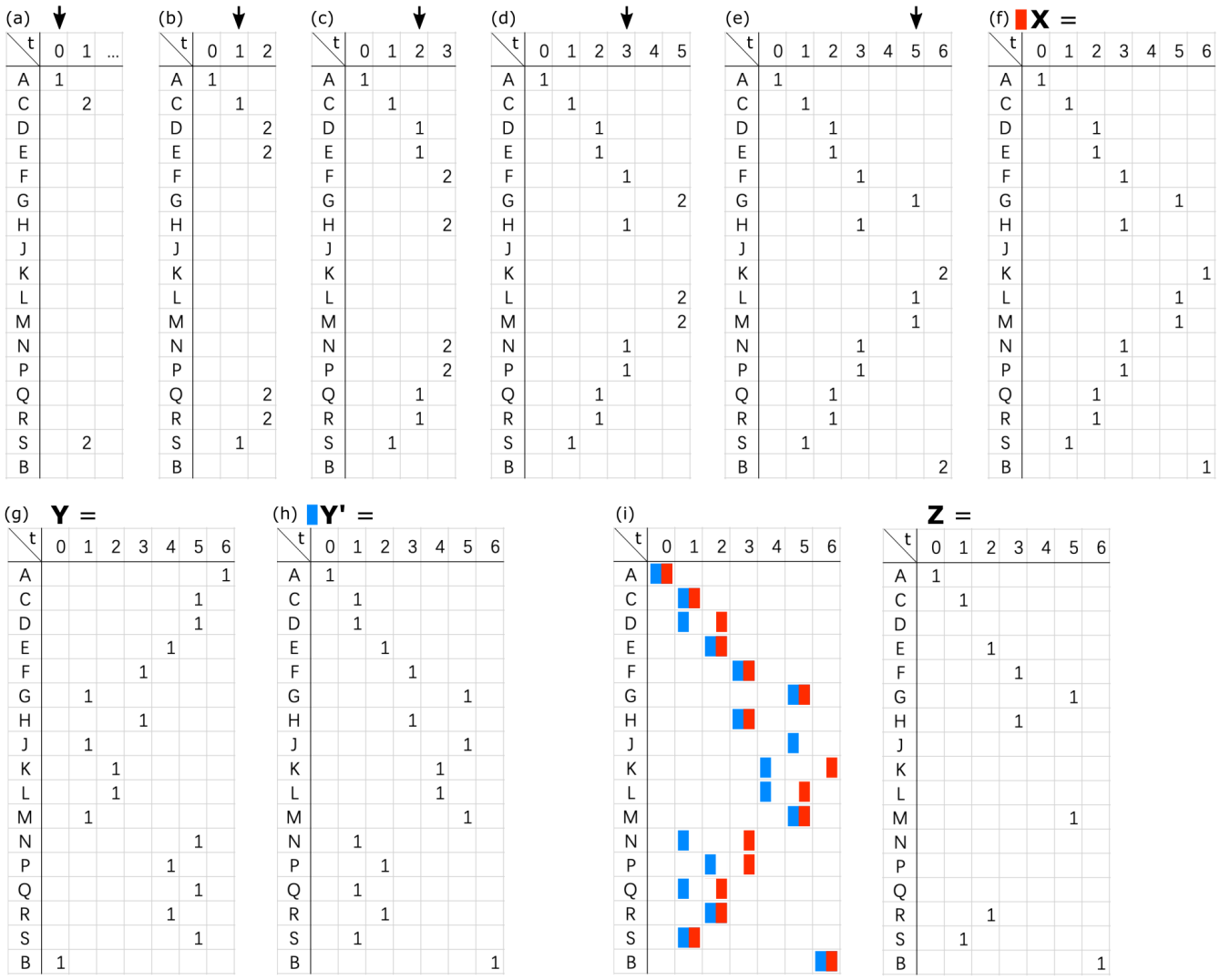


Fig. 2. The matrix implementation of the resonance algorithm. Note that we have omitted to write 0 explicitly in these matrices. (a)-(f) explain how to obtain matrix $\mathbf{X}$ for the forward process. (g) shows the matrix $\mathbf{Y}$ for the backward process. (h) shows the time-reversed matrix $\mathbf{Y}'$. (i) shows how to obtain the final matrix $\mathbf{Z}$. For the red entries, the matrix $\mathbf{X}$ has value 1; while for the blue entries, the matrix $\mathbf{Y}'$ has value 1. The entries with red and blue markers simultaneously (i.e., the forward and backward processes are overlapped at those positions) will be set to value 1 so that we obtain the final matrix $\mathbf{Z}$.

2.3. Find overlap.

- We will reverse the time for the backward process: Create a new matrix $\mathbf{Y}'$ such that $col_j(\mathbf{Y}') = col_{(W-j+1)}(\mathbf{Y})$ where W is the number of columns of $\mathbf{Y}$, as shown in Fig. 2h.

- Now, we will find the overlaps between $\mathbf{X}$ and $\mathbf{Y}'$: Create a matrix $\mathbf{Z}$ such that $Z_{i,j} = X_{i,j} \wedge Y'_{i,j}$, as shown in Fig. 2i.

Finally, the matrix $\mathbf{Z}$ is the one that contains the information of the shortest paths from A to B, from which we can visualize the paths as shown in Fig. 1b.

3. Discussions

The resonance algorithm is very intuitive. The basic idea is that if each node sends signals to all of its neighbors immediately after it receives one from its neighbors, then when the destination receives the signal for the first time, the signal must have traveled through the shortest path to reach the destination. The information about the shortest path has already been stored in this process, but the problem is how to extract it.

On the other hand, if the signal is sent from the destination to the starting point, there must be a signal also traveling through the shortest path. The overlap of the forward process and the backward (in reverse time) process will reveal the shortest paths. I will leave the rigorous proof.

(a)

Operation step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	0	0															
C	∞	1	1														
D	∞	∞	2	2	2												
E	∞	∞	2	2	2	2											
F	∞	∞	∞	∞	∞	3	3	3	3								
G	∞	∞	∞	∞	∞	∞	∞	∞	5	5	5	5	5				
H	∞	∞	∞	∞	∞	∞	∞	3	3	3							
J	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	7	7
K	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	6	6	6	
L	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	5	5	5			
M	∞	∞	∞	∞	∞	∞	∞	∞	6	5	5	5	5	5			
N	∞	∞	∞	∞	∞	∞	3	3	3	3	3						
P	∞	∞	∞	∞	∞	∞	3	3	3	3	3	3					
Q	∞	∞	∞	2	2	2	2										
R	∞	∞	∞	2	2	2	2	2									
S	∞	1	1	1													
B	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	6	6	6	6	6

(b)

t	0	1	2	3	4	5	6	7
A	0							
C		1						
D			2					
E			2					
F				3				
G						5		
H				3				
J								7
K							6	
L						5		
M						5		
N				3				
P				3				
Q			2					
R			2					
S		1						
B							6	

Fig. 3. (a) The matrix implementation of Dijkstra's algorithm for the map shown in Fig. 1a. The corresponding nodes marked in grey means that they are visited after specific operation steps. (b) Rewrite the matrix in (a) in terms of time (which is equivalent to distance) rather than the operation step, to compare with matrix $\mathbf{X}$ in Fig. 2f.

3.1. Comparison with Dijkstra's algorithm. I will only compare the resonance algorithm with the classic Dijkstra's algorithm because other typical algorithms such as Bellman-Ford, A*, Floyd-Warshall, and Johnson's algorithm are all somehow related to Dijkstra's algorithm, which has been intensively studied before [see Wikipedia].

The forward process of the resonance algorithm has a few points in common with Dijkstra's algorithm:

- If a node is currently in focus, the next step is to check all of its neighbors. The resonance algorithm does it by sending signals to all of the neighbors; while Dijkstra's algorithm does it by updating the distances of paths to all of the neighbors.
- If any node has been visited or received signals, ignore them in later researching processes. The resonance algorithm does it by only allowing nodes to send signals if they never sent one before; while Dijkstra's algorithm does it by keeping a list of unvisited nodes.

One of the major differences is that Dijkstra's algorithm always keeps the distances of the current shortest paths (and keep updating them) while the resonance algorithm only records the time points when nodes send signals.

To compare the two algorithms, we now employ Dijkstra's algorithm to work out the shortest path of the exemplified map in Fig. 1a, by using the similar matrix notation as above (see [Wikipedia] for general steps of Dijkstra's algorithm).

- Initialize a matrix where one row represents one node and one column represents one operation step. The entries of the matrix record the *tentative distances* from the starting point to this node. Firstly, we set the entry for the starting point node A to value 0, and other entries to infinity ∞. Mark all nodes as *unvisited* (and *visited* nodes are marked in grey). Refer to the matrix shown in Fig. 3a, which evolves as the algorithm goes (currently only look at column 0).
- Currently, we are at the initial operation step (column 0). Select one unvisited node that is marked with the smallest tentative distance, node A in this case. Consider all of its unvisited neighbors (nodes C and S in this case), and calculate their tentative distances (both are $0 + 1 = 1$ where 0 is the tentative distance of A and 1 is the distance from A to C or to S). As the new tentative distance 1 is smaller than their current value ∞, update the entries to 1 in column 1 (the 1st operation step) and keep other entries the same. Finally, mark node A as visited. Refer to column 1 in Fig. 3a.
- Currently, we are at the 1st operation step (column 1). Select one unvisited node with the smallest tentative distance. Either C or S works, but we choose C and leave S for the next time. Consider all of C's unvisited neighbors (namely, D and E), and calculate their tentative distances (both are $1 + 1 = 2$). As 2 is smaller than their current value ∞, update the entries to 2 in column 2 (the 2nd operation step). Finally, mark node C as visited. Refer to column 2 in Fig. 3a.
- Now we are in column 2. Select one unvisited node with the smallest tentative distance, S in this case. Calculate the tentative distances of all of S's unvisited neighbors (namely, Q and R). Both calculated tentative distances are $1 + 1 = 2 < \infty$. Then update the entries to 2 in column 3. Finally, mark node S as visited. Refer to column 3 in Fig. 3a.
- Now we are in column 3. Select one unvisited node with the smallest tentative distance, D in this case. Calculate the tentative distances of all of D's unvisited neighbors (namely, E). But the calculated tentative distance is $2 + 1 = 3$ which is larger than the current value 2. So we do not update it. Still, we mark node D as visited. Refer to column 4 in Fig. 3a.
- This process continues, until the destination node B is marked as visited, referring to the final matrix in Fig. 3a.

To compare the matrix with the one obtained from the resonance algorithm (Fig. 2f), we rewrite it in terms of time rather than the operation step. Then we obtain the matrix shown in Fig. 3b. We see that the non-empty entries have exactly the same positions as $\mathbf{X}$'s value-1 entries. Therefore, Dijkstra's algorithm and the forward process of the resonance algorithm are essentially very similar.

The difference between the two algorithms comes after we obtain this matrix. The resonance algorithm works out the shortest paths by finding the overlaps between the forward process and the backward (in reverse time) process; while Dijkstra's algorithm does it by revisiting this process and memorizing the shortest paths leads to the destination.

Due to the similar process mentioned above, the two algorithms have similar algorithmic complexity. On one hand, the resonance algorithm requires an extra backward process which may double the running time (but contributes nothing to the algorithmic complexity); On the other hand, it requires neither to memorize the paths visited nor record paths' distances, which may reduce the running time. In practice, various techniques can be applied to achieve better performance.

3.2. Decentralized computing. One important property of the resonance algorithm is that a fully decentralized version is possible since every node can act as an independent agent. Each node has exactly the same rules and its behavior depends only on the local information, i.e., which node it is linked to and when it receives signals.
The following could be a recipe for the decentralized version of the resonance algorithm:

- Each node only records who it is linked to, namely all of its neighbors (so there is no need for a complete map to be stored somewhere);
- The signal any node sends should hold the information about where and when it was sent from.
  When the destination receives signals, the shortest paths can then be reconstructed.

3.3. Thoughts: quantum mechanics & gravitational field. The process that each node sends signals to all of its neighbors when it receives one can be considered as that a signal is experiencing every possible path. As long as the destination is determined, the shortest path (equivalently, the fastest way) is automatically determined. Surprisingly, it seems like the behavior of wave functions in quantum mechanics.

Now we imagine the classic photon single-slit diffraction experiment [see Wikipedia], i.e., a photon passes an extremely small slit to reach a fluorescent screen behind. The photon's wave function expands behind the slit until it reaches the screen. The wave function only tells us the probability of this photon's position, rather than an exact position. But when we see a flash on the screen (namely, the photon is detected), we immediately know that the photon travels through the straight line connecting the slit and the flashing point (which is the shortest path). This might be called "wave function collapse" (one of the many physical interpretations, though), which I found intuitively similar to the resonance algorithm: The signal is experiencing every possible path; but as long as the destination receives the signal, the shortest path automatically reveals itself.

For another example, let us consider the gravity between Earth and the Moon. One way to think about it (my old way) is that a gravitational field is first created by Earth, and the Moon only perceives this gravitational field around itself. This surrounding gravitational field determines the Moon's movement. Another way to think about it (inspired by the resonance algorithm) is to consider Earth and the Moon at the same time. That is, they both create their own gravitational field that expands outwards in all directions in the speed of light; when both fields reach the other object, the interaction forms. This mechanism guarantees that the interaction forms in the quickest way because, as the resonance algorithm, the gravitational field is experiencing every path before the interaction forms. And notice that natural interactions indeed occur in the quickest way.

Are the two examples above suggesting that the resonance-algorithm-like process is the underlying mechanism of how natural interactions form (because it can automatically explain why all of the natural interactions form in the quickest way)? These thoughts are by no means scientifically solid, but it is deserved to be said if it offers any inspiration, reflection or chance of discussions.

ACKNOWLEDGMENTS. The author gives special thanks to Daniel Hjerpe (currently in Ericsson, Sweden) for his detailed comments and enthusiasm on this project, also thanks to Dr. BinBin Hong (currently in Shenzhen University, China) and YiBin Jiang (currently in University of Glasgow, UK) for fruitful discussions.
This work was first finished in February 2013 when the author was studying physics at Sichuan University, China. It was not published in peer-reviewed journals as the author does not know how to evaluate it. Now in April 2020, the author decided to put it on his own personal website.