International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 7, Number 2, pages 349–371, July 2017

A Self-optimizing Routing Algorithm using Local Information in a 3-dimensional Virtual Grid Network with Theoretical and Practical Analysis[1]

Yonghwan KIM
Graduate School of Engineering, Nagoya Institute of Technology
Aichi, 466–8555, Japan
Email: kim@nitech.ac.jp

and

Yoshiaki KATAYAMA
Graduate School of Engineering, Nagoya Institute of Technology
Aichi, 466–8555, Japan
Email: katayama@nitech.ac.jp

Received: February 14, 2017
Revised: April 28, 2017
Accepted: June 2, 2017
Communicated by Akihiro Fujiwara

Abstract

In this paper, we present a self-optimizing routing algorithm using only local information, in a three-dimensional (3D) virtual grid network. A virtual grid network is a well-known network model for its ease of designing algorithms and saving energy consumption. We consider a 3D virtual grid network which is obtained by virtually dividing a network into a set of unit cubes called *cell*s. One specific node named a *router* is decided at each cell, and each router is connected with the routers at adjacent cells. This implies that each router can communicate with 6 routers.

We consider the maintenance of an inter-cell communication path from a source node to a destination node and propose a distributed self-optimizing routing algorithm which transforms an arbitrary given path to an optimal (shortest) one[2] from the source node to the destination node. Our algorithm is executed at each router and uses only local information (6 hops: 3 hops each back and forward along the given path). Our algorithm can work in asynchronous networks without any global coordination among routers.

We present that our algorithm transform any arbitrary path to a shortest path in $O(|P|)$ synchronous rounds, where $|P|$ is the length of the initial path, when it works in synchronous networks. Moreover, our experiments show that our algorithm converges in about $\frac{|P|}{2}$ synchronous rounds and the ratio becomes lower as $|P|$ becomes larger.

Keywords: Routing Algorithm, Self-Optimizing, Local Optimization, Grid Networks

[1]The preliminary version of this paper appeared in the proceedings of the Fourth International Symposium on Computer and Networking (CANDAR) 2016, which entitled *A Self-optimizing Routing Algorithm in a 3-dimensional Virtual Grid Network*.

[2]Note that, in 2D/3D (virtual) grid network, a shortest path between two nodes is not unique.

1 Introduction

Recently, wireless networks, such as MANET (Mobile Ad-hoc NETworks)[2, 3] or WSN (Wireless Sensor Networks)[4], become popular because of widespread use of wireless devices. Therefore a study of the wireless networks becomes important in the distributed systems. In the typical wireless networks, nodes are deployed on a two-dimensional plane, and each node can directly communicate only with nodes within its communication range. If the destination node (the node receives the message) is outside of the communication range of the source node (the node sends the message), the message should be relayed to the destination node. The topology of wireless networks can be changed frequently because of moving of nodes. Multi-hop routing algorithms for wireless networks have been proposed [6, 7, 8, 9, 10, 11].

A virtual grid network is a well-known topology model for designing algorithms of dynamic networks. A typical virtual grid network on a two-dimensional plane, which is obtained by virtually dividing a wireless network into a grid of geographical square regions are called cells of the same size. Figure 1 shows an example of the virtual grid network. Note that the size of the cells is determined by the communication range of each node.

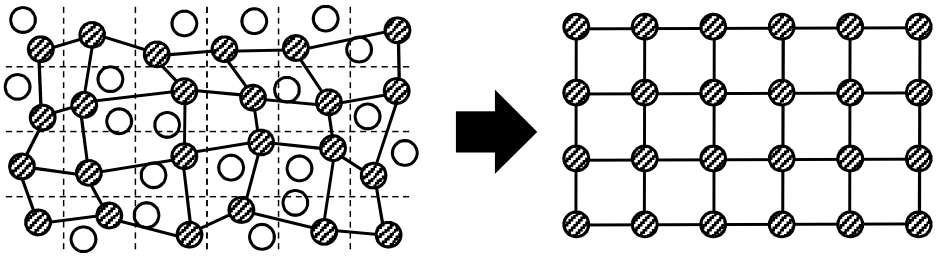


Figure 1: A typical virtual grid network (on 2D plane)

In this paper, we consider a virtual grid network on the three-dimensional (3D) space. Nodes are deployed in the 3D space, each node can communicate with nodes within its communication range same as the case of 2D plane. In a 3D virtual grid network, each cell's shape becomes a unit cube like Figure 2. Each node in the 3D space has a spherical communication range instead of a circular one on a 2D plane and the size of each cell (unit cube) is determined by the communication range of each node. A specific single node called a router is selected at each cell (the marked node in Figure 2) and each router communicates with routers in neighbor cells. Figure 3 illustrates an example of a 3D virtual grid network.

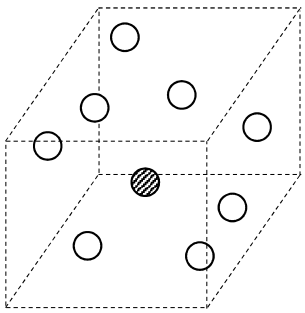


Figure 2: A cell (unit cube) in the virtual grid network

Each node can communicate with other nodes that exist outside of the communication range using routers in a 2D/3D virtual network. We suppose two specific nodes, one named a *source* node and the other named a *destination* node in virtual grid networks, and we allow that each node can

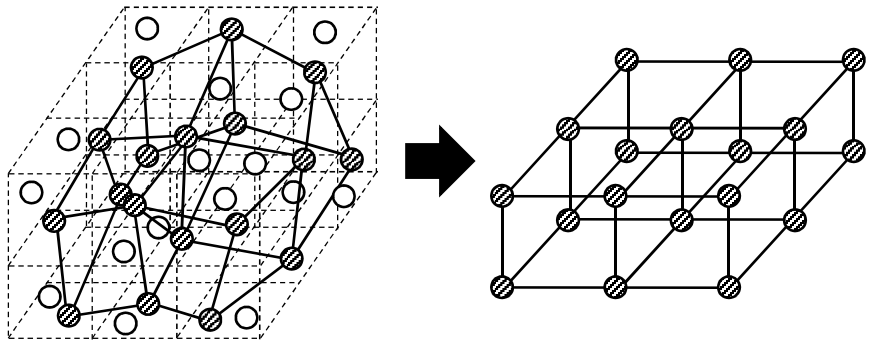


Figure 3: A virtual grid network on the 3D space

move from its current cell to the neighbor cell freely. And we assume that there is a path, which is represented by the sequence of the routers (details will be introduced in Section 3), from a *source* node to a *destination* node. When a node moves to a neighbor cell (there are 6 neighbor cells in a 3D virtual grid network), the path is updated by extending itself to the next (neighbor) cell.

In this paper, we consider the maintenance of the given path from a *source* node to a *destination* node. The power consumption is one of the important issues in wireless sensor networks, therefore, we consider that our goal is to construct a shortest path from a *source* node to a *destination* node. Thus we propose a self-optimizing protocol which can transform an arbitrary given (initial) path from a *source* node to a *destination* node to a shortest path in a 3D virtual grid network.

The rest of this paper is organized as follows: Section 2 presents related works and we introduce the system model we consider and explain the definition of the problem in Section 3. Our proposed self-optimizing algorithm for constructing a shortest path from a source node to a destination node in a 3D virtual grid network is represented in Section 4. We theoretically analyze the time complexity and practically evaluate the convergence time (synchronous rounds) of our proposed algorithm through some experiments in Section 5. Finally we conclude our work and present some future works in Section 6.

2 Related works

Xu et al. proposed a routing algorithm which is named GAF (Geographic Adaptive Fidelity)[5]. GAF reduces energy consumption in ad hoc wireless networks by identifying nodes that is necessary for a routing and turning off unnecessary nodes, keeping a consistent level of routing fidelity. To realize GAF protocol, each node associates itself with a virtual grid so that the entire area is divided into several square regions called cells. When each node has the communication (radio) range R, the length of each side of each grid r is set as $r \leq \frac{R}{\sqrt{5}}$. Herewith all nodes in the cell can communicate with its adjacent cells. In a virtual grid network, a single node is selected as a router and it has responsibility for communication with its adjacent cells for routing and the other nodes can communicate with the router in the same cell when they need to communicate outside of their communication range (when no communication is needed, they can sleep for saving energy). A virtual grid network makes a routing problem simple because a grid architecture is simple and regular.

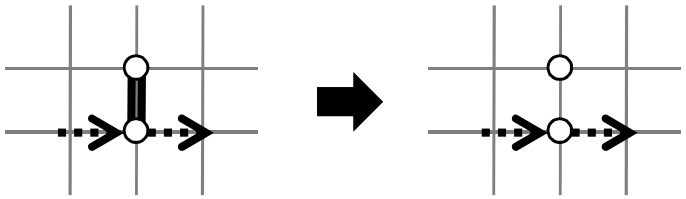
Takatsu et al. proposed a local-information-based self-optimizing routing algorithm in virtual grid networks which is named *Zigzag*[1]. The system model of *Zigzag* is as follows:

- **Asynchronous Network**. Each node in the network can operate freely at any time.
- **No global information**. Each node does not know where a *source* node and a *destination node* are. There is no coordination system, this means that each node does not have any location information.

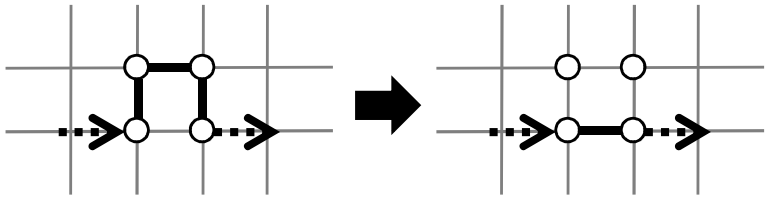
- **Labeled Links**. Each router maintains four communication links because it communicates with four routers in its adjacent cells. Each router labels its four communication links as UP, RIGHT, DOWN, and LEFT. All the routers have a common sense of direction. This means that if a sender sends a message through the link labeled RIGHT, a receiver receives it through the link labeled LEFT.

- **Local information within constant distance**. Each router maintains the subsequence of the routing path that contains 6 hops (directions) which consists of 3 hops back and forward along the path.

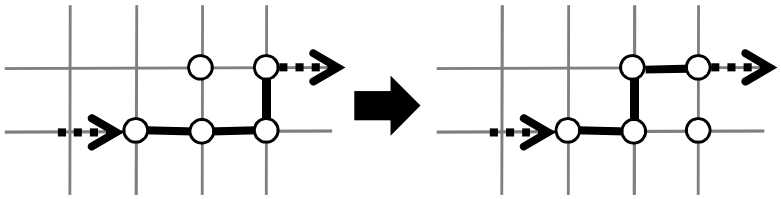
Zigzag uses only three rules to locally update a path. When a router finds a condition of these three rules, it locally update the subsequence of the path around it. Figure 4 simply shows the three rules of *Zigzag*.



(a) *Zigzag* Rule 1 : Shortcut1



(b) *Zigzag* Rule 2 : Shortcut2



(c) *Zigzag* Rule 3 : Zigzag

Figure 4: Three rules of *Zigzag* protocol

Zigzag protocol ensures to construct a shortest path within reasonable convergence time ($O(|P|)$, where $|P|$ is the length of the initial given path) using only these three rules. *Zigzag* also guarantees the preservation of the connection between a *source* node and a *destination* node during the execution of the algorithm.

3 System model and problem definition

In this section, we introduce our system model and define our problem.

3.1 System model

We suppose all nodes in the system are deployed in 3D space. Each node has the sphere-shaped communication range with radius R, and it can communicate with the nodes within R. A 3D virtual grid network is obtained by virtually dividing an entire network into 3D grid (unit cube) cells. One node is selected as a router in each cell, each router can communicate with 6 routers in its adjacent cells. This implies that each vertex has 6 edges in a 3D virtual grid network. If there is no node in a cell, no router exists in the cell. In this case, inter-cell communication among cells cannot be guaranteed, and this causes that any communication path through this non-router cell may not be constructed. Therefore, we assume that at least one node exists in every cell.

There are two nodes, a *source* node v_s and a *destination* node v_t, in the system and we assume that some messages are periodically transferred from v_s to v_t. A message is sent from v_s to the router p_s which is the router in the same cell. The message is relayed by some routers to p_t which is the router in the same cell with v_t. Finally, the message is delivered to v_t. Note that v_s and p_s may be the same node, and v_t and p_t also may be the same. Each non-router node always communicates with the router in the same cell for routing, thus we consider only routers (ignore non-router nodes) in the 3D virtual grid network for ease of designing algorithms. The entire 3D virtual grid network is represented as an undirected graph $N_G = (P_G, E_G)$ (where, P_G is a set of routers and E_G is set of links). A Router p_j is neighbor to p_i, this implies that p_i and p_j share a face of the cell, if and only if a link $(p_i, p_j) \in E_G$.

Each router p_i has 6 links because all cells are cube-shaped in a 3D virtual grid network. Each router has no knowledge of its global location, e.g. (x, y, z)-coordinates, however all the routers have a common sense of 6 directions, and they are labeled on these 6 links of each router. Figure 5 shows labels of all 6 links of p_i. Each router has 6 links which are labeled with $\mathcal{T}$(Top), $\mathcal{B}$(Bottom), $\mathcal{U}$(Up), $\mathcal{D}$(Down), $\mathcal{L}$(Left) or $\mathcal{R}$(Right) consistently. For example, if a router sends a message through the link labeled $\mathcal{R}$, a receiving router receives the message through the link labeled $\mathcal{L}$, because of common sense of direction.

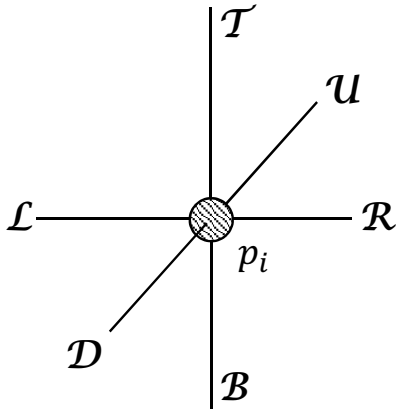


Figure 5: A direction label of each link of the router p_i

A routing path P to a *destination* node p_t from a *source* node p_s can be represented by the sequence of routers as follows: $P = (p_s = p_0, p_1, p_2, \ldots, p_n = p_t)$. In this case, P consists of $(n+1)$ routers and the length of P becomes n. We represent the length of the routing path with $|P|$, therefore $|P| = n$ implies that n-hops are required to deliver a message from p_s to p_t and $(n-1)$ routers exist between p_s and p_t. Due to common sense of direction, each edge in P can be represented by a direction label. For example, if p_i sends a message through its link labeled $\mathcal{R}$, $p_{(i+1)}$ receives it through its link labeled $\mathcal{L}$ inevitably. Therefore we can represent the edge of P between p_i and $p_{(i+1)}$ as a single direction label $\mathcal{R}$ which is p_i's outgoing link's direction. Thereafter P also can be represented by the sequence of edges or direction labels as follows: $P =$

$(\overrightarrow{p_0,p_1}, \overrightarrow{p_1,p_2}, \overrightarrow{p_2,p_3}, \ldots, \overrightarrow{p_{(n-1)},p_n}) = (Out(p_0), Out(p_1), Out(p_2), \ldots, Out(p_{(n-1)})$ (where $Out(p_i)$ is the direction label of p_i's output edge).

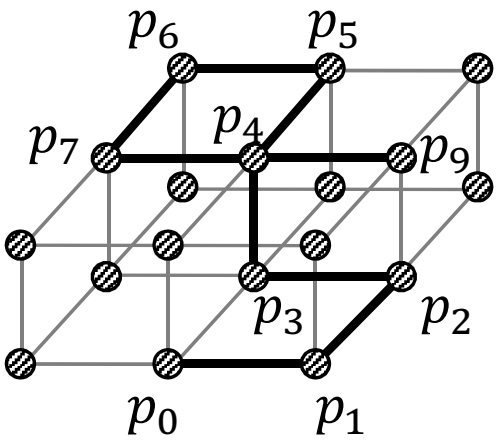


Figure 6: An example of a routing path P

Figure 6 shows an example of the representation of P. In the case of Figure 6, $|P|$ consists of 10 routers, therefore $|P| = 9$. Note that the router p_4 and p_8 are considered as the different routers even though p_4 and p_8 are the same router. P can be represented by the sequence of the direction labels as follows: $P = (\mathcal{R},\mathcal{U},\mathcal{L},\mathcal{T},\mathcal{U},\mathcal{L},\mathcal{D},\mathcal{R},\mathcal{R})$. In this paper, we mainly use this notation (using direction labels) unless specifically mentioned. In addition, we use the notation D_i which means the direction label of p_i's output edge $Out(p_i)$.

Each router which belongs to the routing path from p_s to p_t maintains its routing table. A routing table consists of some records which is notated by the pair of input direction and output direction like (In, Out). For instance, p_2 in Figure 6 has the record $(\mathcal{D},\mathcal{L})$ in its routing table because p_2 should forward a message received through the link labeled $\mathcal{D}$ to the link labeled $\mathcal{L}$ in order to transfer a message from a *source* node to a *destination* node. $p_4(= p_8)$ stores two records in its routing table, $(\mathcal{B},\mathcal{U})$ for p_4 and $(\mathcal{L},\mathcal{R})$ for p_8.

We assume that each router knows routing records of preceding and following 2 routers (total 6 hops). For example, a router p_i maintains additional routing records of $p_{(i-2)}$, $p_{(i-1)}$, $p_{(i+1)}$, and $p_{(i+2)}$. A router p_i maintains P^{p_i} which is the subsequence of the current routing path P, P^{p_i} = ($Out(p_{(i-3)})$, $Out(p_{(i-2)})$, $Out(p_{(i-1)})$, $Out(p_i)$, $Out(p_{(i+1)})$, $Out(p_{(i+2)})$) and $|P^{p_i}|$ becomes 6 obviously (except some routers like p_s or p_t). Note that $Out(p_{(i-3)})$ can be derived from $p_{(i-2)}$'s routing record.

3.2 Problem definition

A valid path P from p_s to p_t is initially given in a 3D virtual grid network. We say P is valid if and only if $P = (p_0, p_1, p_2, \ldots, p_n)$ fulfills the following conditions: (1) p_0 is the router in the same cell with v_s (i.e. $p_0 = p_s$), (2) p_n is the router in the same cell with v_t (i.e. $p_n = p_t$), and (3) Each p_i (except p_0 and p_n) is the router in the neighboring cell of $p_{(i-1)}$ and $p_{(i+1)}$. These conditions guarantee the connectivity of the path P.

The path optimization problem is, from any given initial valid path P in the 3D virtual grid network N_G, to construct a shortest path from p_s to p_t in N_G.

4 Our proposed algorithm

In this section, we introduce our algorithm which solves the path optimization problem, using each router's local information only.

In the previous section, we introduced the notation of P using each router's direction label: $P = (Out(p_0), Out(p_1), Out(p_2), \ldots, Out(p_{(n-1)}))$.

Now we define an operation "$\star$" of 6 directions $D_i = \{\mathcal{U}, \mathcal{D}, \mathcal{R}, \mathcal{L}, \mathcal{T}, \mathcal{B}\}$ (see Figure 5) as follows. For ease of explanation, we call directions $\{\mathcal{U}, \mathcal{D}, \mathcal{R}, \mathcal{L}\}$ the *horizontal directions*, and directions $\{\mathcal{T}, \mathcal{B}\}$ the *vertical directions*. Note that this operation is not commutative.

1. **Straight Line** : When D_j is the same direction as D_i, $D_i \star D_j = 1$ (e.g. $D_i = D_j = \mathcal{R}$).
2. **Retrace** : When D_j is the opposite direction of D_i, $D_i \star D_j = -1$ (e.g. $D_i = \mathcal{U}$ and $D_j = \mathcal{D}$).
3. **Bend on a Plane** : When both D_i and D_j are the *horizontal directions* and they are orthogonal, $D_i \star D_j = 0$ (e.g. $D_i = \mathcal{L}$ and $D_j = \mathcal{U}$).
4. **Horizontal Bend** : When D_i is the *vertical direction* and D_j is the *horizontal direction*, $D_i \star D_j = 2$ (e.g. $D_i = \mathcal{T}$ and $D_j = \mathcal{U}$).
5. **Null OP** : In the other case than listed above, $D_i \star D_j = \emptyset$. (e.g. $D_i = \mathcal{R}$ and $D_j = \mathcal{B}$).

This operation "$\star$" is basically the same as inner product of two vectors when both vectors are on a plane surface. However in the case of the bending from vertical direction to horizontal direction, we assign a new value 2 in our algorithm. Now we introduce our local update rules in the following subsection.

4.1 Local update rules

Let $P = (D_0, D_1, D_2, \ldots, D_{(n-1)})$ be the path to p_t from p_s (D_i means $Out(p_i)$). We define four rules of the local update on each p_i from some i as follows.

1. **Shortcut1** : If $D_{(i-1)} \star D_i = -1$, $D_{(i-1)}$ and D_i are removed from P.
2. **Shortcut2** : If $D_{(i-1)} \star D_i = 0$ and $D_{(i-1)} \star D_{(i+1)} = -1$, $D_{(i-1)}$ and $D_{(i+1)}$ are removed from P.
3. **ZigZag** : If $D_{(i-1)} \star D_i = 1$ and $D_i \star D_{(i+1)} = 0$, D_i and $D_{(i+1)}$ are exchanged their sequence in P.
4. **V-Shift** : If $D_{(i-1)} \star D_i = 2$, $D_{(i-1)}$ and D_i are exchanged their sequence in P.

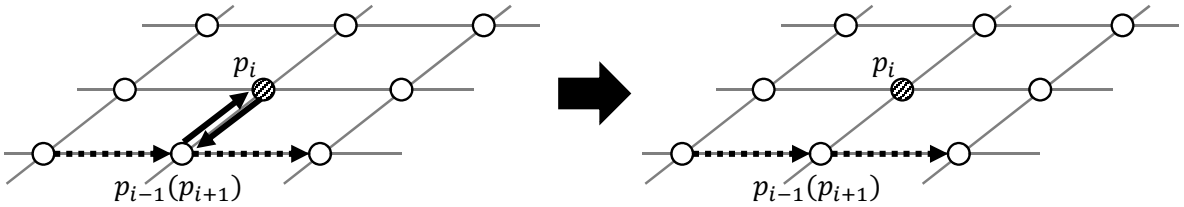
Figure 7 shows the examples of four local update rules of our proposed algorithm. Note that *Shortcut2* and *ZigZag* are updated on a plane surface only. Each router p_i always checks the relation between $p_{(i-1)}$'s output direction ,which is the counter direction of p_i's input direction, and its output direction (i.e. $D_{(i-1)} \star D_i$). Figure 8 represents the flow chart of checking local update rules of p_i.

In this paper, we abbreviate how each local update is implemented. We consider each local update can be executed pseudo-atomically. To help to design atomicity, our algorithm ensures that an edge is never targeted by two or more local updates. This implies that each local update is spared the intrusion of the other local updates. Details will be mentioned in the next subsection.

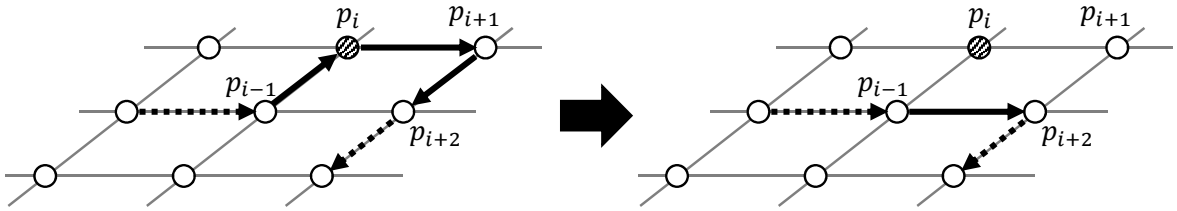
4.2 Collision handling of local updates

Some edges may be targeted for two local updates at the same time. Figure 9 shows an example of some local update rules' collision.

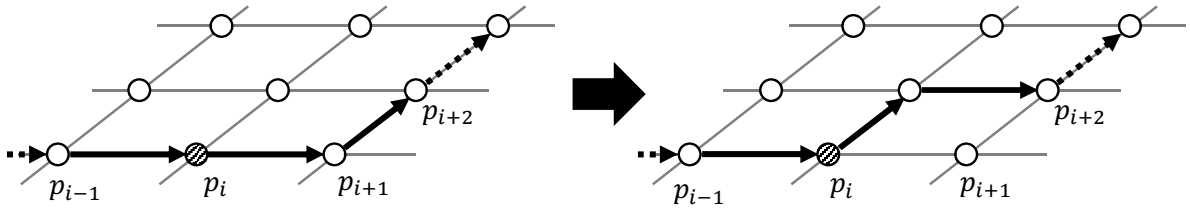
In Figure 9, p_i detects the local update *V-Shift*, $p_{(i+1)}$ detects the local update *Shortcut1*, and $p_{(i+2)}$ detects the local update *Shortcut2* at the same time. In this case, some edges may be removed or moved (exchanging its sequence) by two local updates at the same time. For example, D_i is moved by p_i's local update (*V-Shift*) and removed by $p_{(i+1)}$'s local update (*Shortcut1*) simultaneously. On



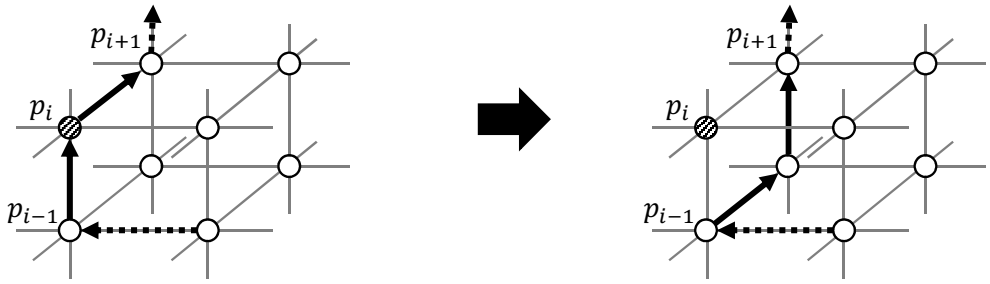
(a) Shortcut1



(b) Shortcut2



(c) ZigZag



(d) V-Shift

Figure 7: Four rules of our proposed algorithm

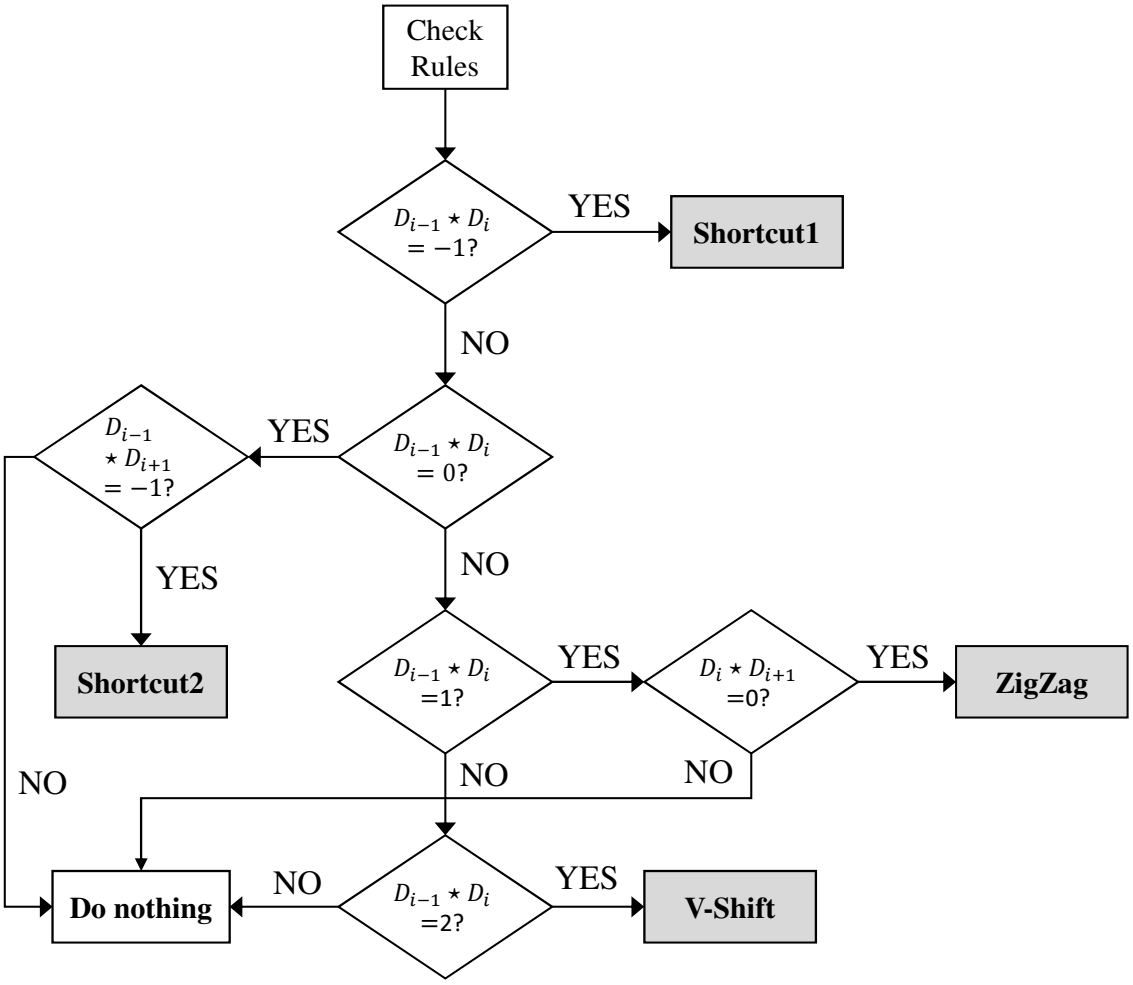


Figure 8: Flowchart of checking local update rules

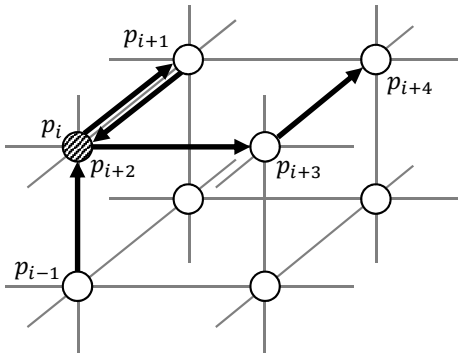


Figure 9: Some local update rules' collision

the other hand, $D_{(i+1)}$ is removed by $p_{(i+1)}$'s local update (*Shortcut1*) and removed by $p_{(i+2)}$'s local update (*Shortcut2*) simultaneously.

These collisions (conflictions), which mean the situation of which some edges are targeted by two or more local updates, may cause the disconnection of the routing path to p_t from p_s. Therefore, some exclusion rules are required to avoid disconnection of the path. A router p_i never detects two or more local update rules at the same time because the definition of local update rules. As presented in Figure 8, a router p_i detects one local update rule or not (do nothing) at some instant. Hence we only consider two different routers p_i and p_j trying to remove (or move) the same edge. To realize the mutual exclusion of local update rules, we set one simple rule: When a router p_i detects a local update rules, p_i checks some other update rules including $D_{(i-1)}$. If p_i detects some other local update rules including D_i, p_i ignores its local update rule detected. Figure 10 shows some examples of the collision (confliction) between two local update rules. In Figure 10(a), p_i detects a local update *Shortcut1* because of $D_{(i-1)} = \mathcal{U}$ and $D_i = \mathcal{D}$. However, p_i knows $D_{(i-2)} = \mathcal{D}$ and it finds that $p_{(i-1)}$ detects a local update *Shortcut1*. Thus, p_i does not execute its local update. In a similar fashion, p_i in Figure 10(b) and p_i in Figure 10(c) do not execute their local update neither.

Algorithm 1 represents the pseudocodes of our proposed algorithm with a confliction handling.

Algorithm 1 Pseudocodes for p_i: Local-information-based routing algorithm in a 3D virtual grid network

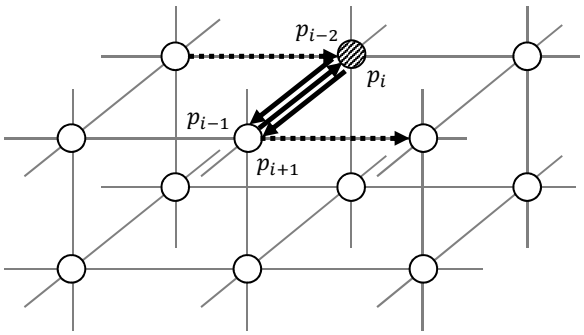
Require: Subset of $P^{p_i} = (D_{(i-3)}, D_{(i-2)}, \ldots, D_{(i+2)})$
Ensure: Local update type of p_i

```
1:  procedure CheckLocal(p_i)
2:      var update ← ⊥
3:      if D_(i−1) ⋆ D_i = −1 then
4:          update ← Shortcut1
5:      else if D_(i−1) ⋆ D_i = 0 & D_(i−1) ⋆ D_(i+1) = −1 then
6:          update ← Shortcut2
7:      else if D_(i−1) ⋆ D_i = 1 & D_i ⋆ D_(i+1) = 0 then
8:          update ← ZigZag
9:      else D_(i−1) ⋆ D_i = 2
10:         update ← V − Shift
11:     end if
12:     if CheckLocal(p_(i−1)) ≠ ⊥ then
13:         update ← ⊥
14:     else if CheckLocal(p_(i−2)) = Shortcut2
15:           | CheckLocal(p_(i−2)) = ZigZag then
16:         update ← ⊥
17:     end if
18:     if update ≠ ⊥ then
19:         Execute Local Update
20:     end if
21: end procedure
```

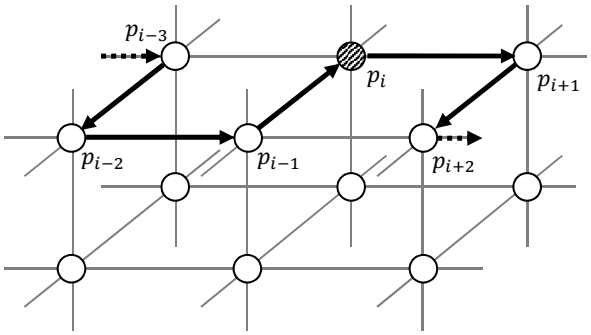
Figure 11 show an example of the execution of our proposed algorithm. Our algorithm can be operated asynchronously, but this example shows the synchronous execution (each router which detects a local update rule executes it at the same time) in order to help to understand.

The initial given path $P = (\mathcal{R}, \mathcal{R}, \mathcal{U}, \mathcal{T}, \mathcal{D}, \mathcal{L}, \mathcal{U}, \mathcal{U}, \mathcal{L}, \mathcal{D})$ is illustrated in Figure 11(a). And three gray routers in Figure 11(a) detect local update rules by Algorithm 1. p_1, p_4 and p_7 detect *ZigZag*, *V-Shift* and *ZigZag*, respectively. Note that p_5 ignores its local update *Shortcut2* because p_5 knows p_4 will execute the local update *V-Shift* from its local-information. Likewise, p_8 also ignores its local update.

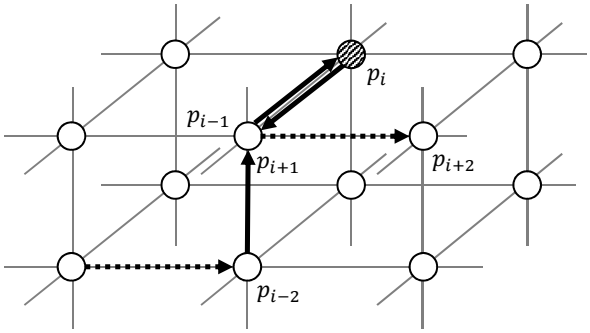
Figure 11(b) shows the updated path after the executions of three gray routers. And three new gray routers in Figure 11(b), p_2, p_5 and p_9, execute their local updates.



(a) Shortcut1 precedes Shortcut1

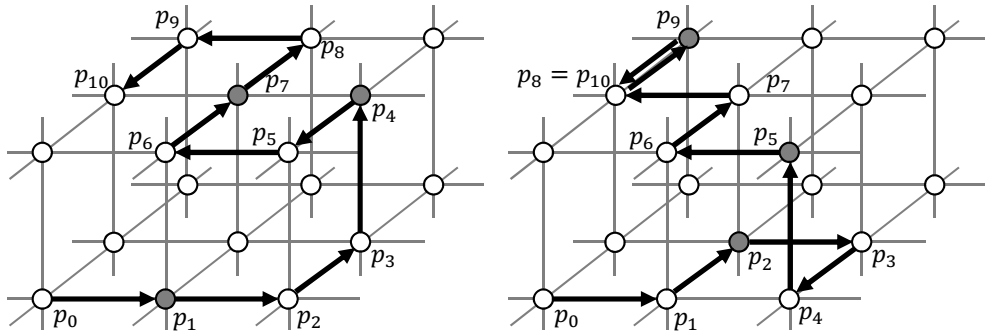


(b) Shortcut2 precedes Shortcut2



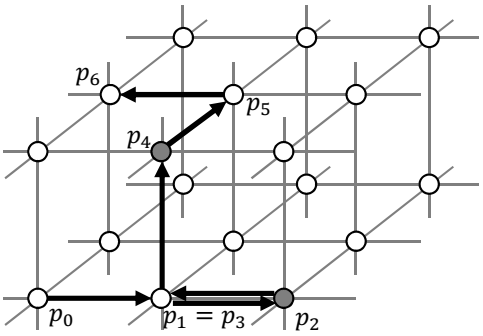
(c) V-Shift precedes Shortcut1

Figure 10: Examples of confliction between two local updates

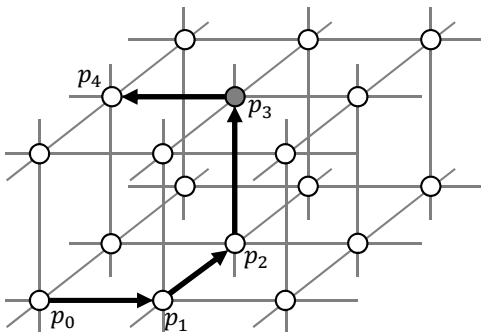


(a) Initial Path

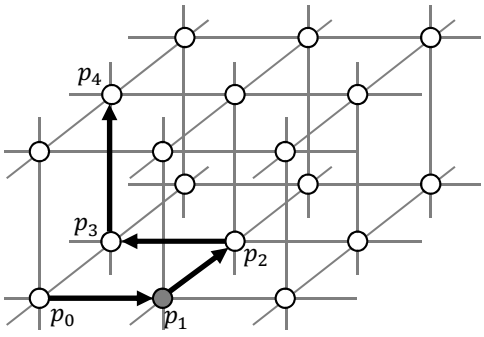
(b) Step 1



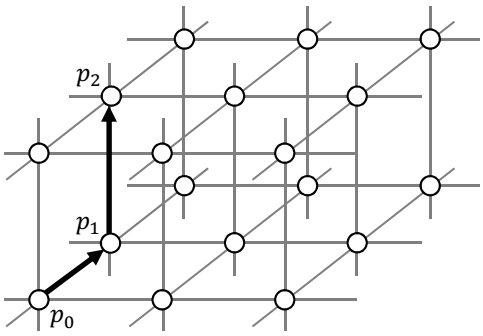
(c) Step 2



(d) Step 3



(e) Step 4



(f) Final Path

Figure 11: Examples of executions of our algorithm

Finally, our algorithm constructs a shortest path like Figure 11(f) and is terminated. This implies that there is no more local update in this path.

To help to understand, in this example, we assume that the destination node is on above side of the source node in the initial given path. That is, it is the same assumption when we described the local rule *V-Shift* in Section 4.1 (Figure 7(d)). However, four local update rules of our proposed algorithm ensure to construct a shortest path correctly regardless of the location of the destination node. We present the correctness of our algorithm in the next section.

4.3 Correctness of proposed algorithm

In this section, we discuss the correctness of our proposed algorithm briefly. We introduce some definitions and we simply give an explanation that our proposed algorithm can construct a shortest path from p_s to p_t.

As we mentioned in the previous section, a routing path P can be represented by the sequence of each routers output link's direction like $P = (D_0, D_1, \ldots, D_{(n-1)})$ where $D_i = Out(p_i)$. Let P be a path from p_s to p_t and assume two (relaying) routers p_a and p_b in P $(0 \leq a \leq b \leq n)$. $\overline{P}$ can be divided into 3 parts, the path from $p_0 (= p_s)$ to p_a (meaning $(D_0, D_1, \ldots, D_{(a-1)})$), from p_a to p_b, and from p_b to $p_n (= p_t)$. We notate these parts P^{sa}, P^{ab}, and P^{bt} respectively. Certainly, P^{sa}, P^{ab}, or P^{bt} can be an empty sequence.

Now we define the converged path $\overline{P}$ which is constructed by our proposed algorithm.

Definition 1. Converged Path $\overline{P}$ of our algorithm. *$\overline{P}$ is the converged path if and only if:*

1. *P^{sa} is alternating with two horizontal directions D_i and D_j, where $D_i \star D_j = 0$.*
2. *P^{ab} consists of all the same horizontal direction (a horizontal line) which appears in P^{sa}.*
3. *P^{bt} consists of all the same vertical direction (a vertical line).*

Figure 12 shows an example of $\overline{P}$. p_b becomes the orthogonal projection onto the same plane with p_s due to its definition.

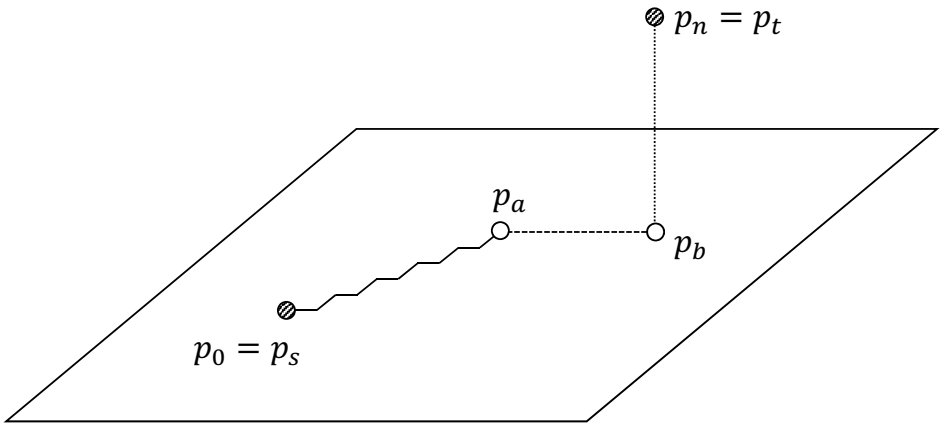


Figure 12: An example of converged path

Lemma 1. *No local update will be executed in converged path $\overline{P}$.*

Proof. At first, at any i $(1 \leq i < a)$, all $D_{(i-1)} \star D_i$ becomes 0 in P^{sa}. However, $D_{(i-1)} \star D_{(i+1)}$ will never be -1 in P^{sa}. Hence there is no local update in P^{sa} of $\overline{P}$.

Secondly, at any i $((a+1) \leq i < b)$, all $D_{(i-1)} \star D_i$ becomes 1 in P^{sa} because P^{ab} consists of only one direction. Thus there is no local update in P^{ab} of $\overline{P}$.

Finally, there is no local update in P^{bt} of $\overline{P}$ due to the same reason of P^{ab}. □

To prove that our algorithm constructs a converged path, firstly we present the following Lemmas 2 and 3.

Lemma 2. *If there is one or more vertical directions which precede any horizontal directions, one or more local updates will be executed.*

Proof. Assume a *vertical* direction D_i is included in P, which is the last *vertical* direction preceding some *horizontal* directions. This implies $D_{(i+1)}$ is a *horizontal* direction. Because of Algorithm 1 (line 9–10), a local update *V-Shift* is detected by $p_{(i+1)}$. Even though there is some local updates preceding $p_{(i+1)}$, *V-Shift* is eventually executed after the other local updates are terminated. □

Lemma 3. *If both T and B are included in P, one or more local updates will be executed.*

Proof. From Lemma 2, no *vertical* directions preceding any *horizontal* directions. We consider that D_i is the last *horizontal* direction in P, thus, $D_{(i+1)}$ becomes the first *vertical* direction in P. From the presumption of Lemma 3, there is D_j in P, where $(i+1) < j \leq n$ and $D_{(i+1)} \neq D_j$. In this case, p_j detects a local update *Shortcut1* because of $D_{(j-1)} \star D_j = -1$.

Therefore, if there is no more local update's execution, either T or B, but not both, is included in P. (or neither of them). □

From Lemmas 2 and 3, we have the following corollary.

Corollary 1. *If no more local update is executed in P, the subsequence $P^{bt} = (p_b, \ldots, p_n) = (D_b, \ldots, D_{(n-1)})$ (where $b \leq n$), which consists of either $\mathcal{T}$ or $\mathcal{B}$, can be determined in P (D_b is the first vertical direction in P).*

From Corollary 1, p_b can be obtained by the orthogonal projection of p_n onto the same plane with p_s (refer to Figure 11).

To make it easier to discuss the convergence of our algorithm, now we only consider the subsequence of P, $P^{sb} = (p_s(= p_0), \ldots, p_b) = (D_0, \ldots, D_{(b-1)})$ which consists of only *horizontal* directions.

Lemma 4. *If 3 or more kinds of directions are included in P^{sb}, one or more local updates will be executed.*

Proof. Assume D_i is the first (*horizontal*) direction with two kinds of different preceding directions in P^{sb}. We show that p_i detects a local updates and eventually executes it.

As the assumption, $D_{(i-1)}$ has the different directions with D_i. If $D_{(i-1)} \star D_i = -1$, p_i detects a local update *Shortcut*1. Thus, we consider $D_{(i-1)} \star D_i = 0$.

In this case, $D_{(i-2)}$ has also different direction with D_i. If $D_{(i-2)}$ is the same direction as $D_{(i-1)}$, a local update *ZigZag* is detected by p_i. If $D_{(i-2)}$ is the opposite direction as D_i, p_i detects a local update *Shortcut2*. If $D_{(i-2)}$ is the opposite direction as $D_{(i-1)}$, a local update *Shortcut1* is detected by $p_{(i-1)}$.

Therefore, if two kinds of different *horizontal* directions precede D_i, $p_{(i-1)}$ or p_i detects a local updates thus Lemma 4 hold. □

From Lemma 4, if two kinds of directions are in P^{sb} and no local update is detected, the directions are orthogonal. Because if not, a local update *Shortcut*1 is detected.

Before the introduction of the next lemma, we define a segment representation of P as follows.

Definition 2. Segment representation of P. *A segment representation of P is expressed by the subsequence of P. Each subsequence S_i of P consists of the consecutive same directions. $SR(P)$ is a segment representation of P, and becomes $SR(P) = (S_1, S_2, \ldots, S_m)$ $(1 \leq m \leq (n-1))$.*

A segment representation combines the consecutive same directions in P into a subsequence of P named S_i. For example, if $P = (\mathcal{U}, \mathcal{U}, \mathcal{D}, \mathcal{L}, \mathcal{T}, \mathcal{T}, \mathcal{T}, \mathcal{R})$, the segment representation of P becomes $SR(P) = ((\mathcal{U}, \mathcal{U}), (\mathcal{D}), (\mathcal{L}), (\mathcal{T}, \mathcal{T}, \mathcal{T}), (\mathcal{R}))$. In this example, there are five segments in P ($m = 5$), and lengths of segments are $|S_1| = 2$, $|S_2| = |S_3| = |S_5| = 1$, and $|S_4| = 3$.

Lemma 5. *When a segment representation of $P^{sb} = (S_0, S_1, \ldots, S_m)$, unless $|S_i| = 1$ at any i $(0 \leq i < m)$, one or more local updates will be executed.*

Proof. Lemma 5 implies that only S_m (the last segment of P^{sb}) has a length of two or more ($|S_m| \geq 2$).

Assume $|S_i| > 1$ $(0 \leq i < m)$ in P^{sb} and no more update is detected in P^{sb}. There is $S_{(i+1)}$, and the last direction of S_i and the first direction $S_{(i+1)}$ become orthogonal (from Lemma 4). In this case, because $|S_i| > 1$, the last two directions of S_i have the same directions. This causes the first router to detect a local update *ZigZag* and this is a contradiction. □

From Lemmas 4 and 5, we obtain the following corollary.

Corollary 2. *If no more local update is executed in P, which consists of only horizontal directions, P includes only two directions which are orthogonal. Moreover, $|S_i| = 1$ at any i $(0 \leq i < m)$ holds in the segment representations of $P = (S_0, S_1, \ldots, S_m)$.*

From Corollaries 1 and 2, unless P is a converged path, our algorithm retains to detect local updates and transforms P. And if P is a converged path, our algorithm stops (is terminated). Now we show our algorithm eventually makes P a converged path, this implies that the infinite executions of our algorithm never occurs. To prove this, we introduce the following definitions using a segment representation.

Definition 3. Potential function f. *Let P be a path and its segment representation $SR(P)$ is $(S_1, S_2, \ldots, S_m)$. The potential function f assigns a sequence $f(P) = (|S_{total}|, |S_1|, |S_2|, \ldots, |S_m|)$, where $|S_{total}| = \sum_{i=1}^{m} |S_i|$.*

Definition 4. Weighted Potential function f^w. *Let $SR(P) = (S_1, S_2, \ldots, S_m)$ be a segment representation of P. The weighted potential function f^w assigns a sequence $f^w(P)$ which is obtained by changing $f(P)$ with the follow rules: If a segment S_i consists of vertical directions (T or B) in $SR(P)$, a value α (defined later) is added to the length of S_i ($|S_i|$) in $f^w(P)$.*

To help to understand $f(P)$ and $f^w(P)$, we present the examples of these functions. In the above case $P = (\mathcal{U}, \mathcal{U}, \mathcal{D}, \mathcal{L}, \mathcal{T}, \mathcal{T}, \mathcal{T}, \mathcal{R})$, $f(P)$ becomes $(|S_{total}|, |(\mathcal{U}, \mathcal{U})|, |(\mathcal{D})|, |(\mathcal{L})|, |(\mathcal{T}, \mathcal{T}, \mathcal{T})|, |(\mathcal{R})|) = (8, 2, 1, 1, 3, 1)$. Note that the first element of $f(P)$ is equal to the length of the path P. However, the weighted potential function $f^w(P)$ becomes $((8 + \alpha), 2, 1, 1, (3 + \alpha), 1)$, because S_4 of $SR(P)$ consists of *vertical* directions T.

We consider that the sequences constructed by potential function are totally ordered by the lexicographic order. In the case of the weighted potential function, we assume that α is large enough (e.g. $\alpha > |P|$). And we call the sequence, constructed by (weighted) potential function, (weighted) potential value.

Now we show that the weighted potential value becomes the minimum value when P is the converged path. And our algorithm eventually decreases the weighted potential value of the given P.

Lemma 6. *If P is a converged path, $f^w(P)$ has the minimum value.*

Proof. Obviously, the minimum weighted potential value is $(|P^{min}|, 1, 1, \ldots, m)$, where $|P^{min}|$ is the length of the shortest path from p_s to p_t and $m \geq 1$ when there is no *vertical* directions in P.

In the weighted potential function, each *vertical* segment is weighted by α, which is a large value, thus it is important to reduce the number of the *vertical* segments for constructing the minimum weighted potential value, that is, there is only one *vertical* segment in $SR(P)$ when the minimum $f^w(P)$. More specifically, the *vertical* segment is positioned as the last segment of $SR(P)$.

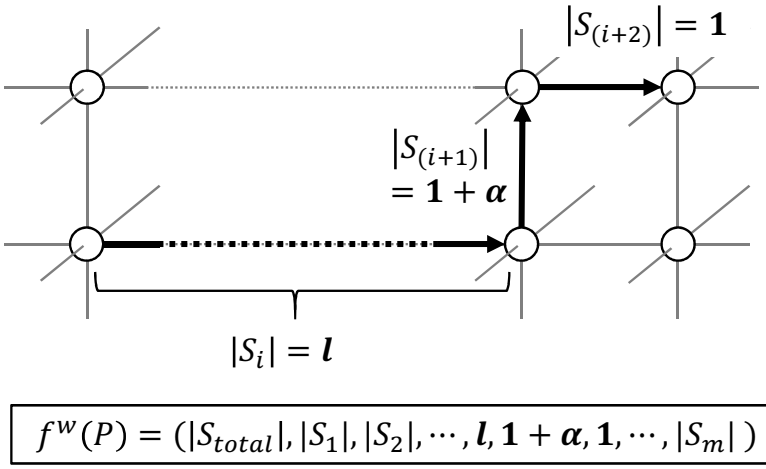
As a result, the minimum weighted potential value becomes $(|P^{min}| + \alpha, 1, 1, \ldots, 1, m, v + \alpha)$, *vertical* directions are included the last segment only. And this becomes a converged path. □

Lemma 7. *The weighted potential value is eventually decreased by our algorithm.*

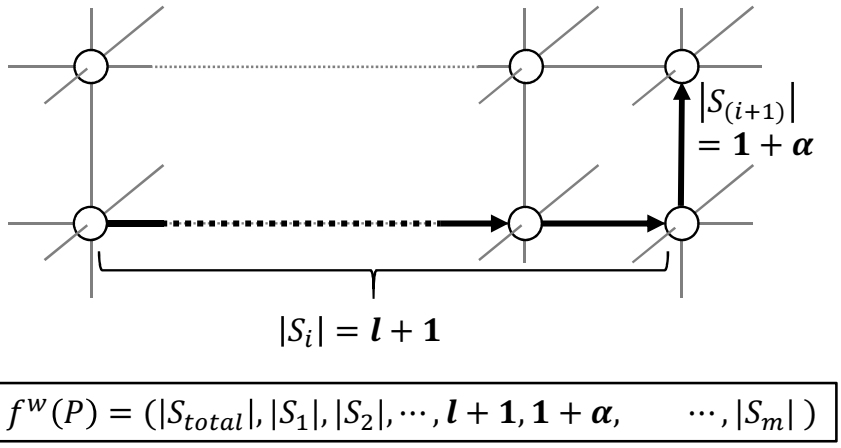
Proof. Local updates *Shortcut1* and *Shortcut2* decrease the length of the given P, thus, the weighted potential value is decreased because the first element of $f^w(P)$ decreases.

Secondly we consider the local update *ZigZag*. If *ZigZag* is executed by the router p_i, D_i (output direction of p_i) is included in S_i of $SR(P)$. In this case, $|S_i| \geq 2$ due to $D_i \star D_{(i-1)} = 1$, and $|S_{(i+1)}| \geq 1$. After executing *ZigZag* on p_i, S_i is decreased by 1. Hence, $f^w(P)$ decreases.

However, local update *V-Shift* sometimes increases $f^w(P)$. For example, if $|S_i| = l$, $|S_{(i+1)}| = 1 + \alpha$ (*vertical* direction), and $|S_{(i+2)}| = 1$ of $SR(P)$ (refer to Figure 13(a)), local update *V-Shift* transforms it to $|S_i| = l + 1$ and $|S_{(i+1)}| = 1 + \alpha$ (refer to Figure 13(b)). This causes the increasing of $f^w(P)$. Though local update *V-Shift* moves each *vertical* direction to backward of the sequence $SR(P)$, and this shift is required for constructing the converged path to minimize the weighted potential value (Lemma 6). There is no local updates moving *vertical* directions to forward of the sequence $SR(P)$, thus, local update *V-Shift* is executed finite number of times (no infinite execution). □



(a)



(b)

Figure 13: Example of increasing the weighted potential value

Our algorithm continues to transform P, which is not converged path, using local update rules, and it convergs after the construction of the converged path $\overline{P}$ within finite executions of local updates (no livelock is guaranteed by Lemma 7).

We obtain the following theorem using Lemmas 1–7 and Corollaries 1 and 2.

Theorem 1. *If P is not a converged path, one or more local updates will be executed in P. And our algorithm constructs the converged path from P within finite number of executions.*

Finally, we show the converged path $\overline{P}$ is a shortest path from p_s to p_t using the following Lemma.

Lemma 8. *A converged path $\overline{P}$ ensures a shortest path from p_s to p_t.*

Proof. We can determine p_b which has the first *vertical* directions as D_b in $\overline{P}$. p_b becomes the orthogonal projection onto the same plane with p_0, $P^{bt} = (p_b, p_{(b+1)}, \ldots, p_t)$ becomes a shortest path from p_t to the plane.

From Lemma 4, $P^{sb} = (p_s(= p_0), p_1, \ldots, p_b)$ consists of two directions which are orthogonal, hence there is no redundancy path in P^{sb}. This also implies a shortest path from p_s to p_b.

Therefore a converged path $\overline{P}$ guarantees a shortest path. □

5 Theoretical analysis and experimental evaluation

In this section, we analyze the time complexity of our proposed algorithm. Moreover, we present our experimental results to show the practical performance of our algorithm.

5.1 Time complexity of our algorithm

In this subsection, we theoretically analyze our algorithm. Note that we evaluate the time complexity of our algorithm when our algorithm is executed on the synchronous environment. Our proposed algorithm correctly works even in asynchronous environment, however, in asynchronous environment (e.g. asynchronous networks), it is impossible to predict the timing of each node's execution, thus, in order to compute the time complexity of our algorithm, we assume that our algorithm works synchronously and we evaluate the time complexity of synchronous executions (rounds). This means that each node detected an enabled local update (refer Figure 9 or Algorithm 1) always executes its local update at each synchronous round.

We also assume that each local update works atomically: each update of the path is executed instantaneously. In our algorithm, the set of routers which detect the valid (without any conflictions) local update are determined deterministically for any given path, thus, the other routers than this set of routers never execute the local update until the one or more updates of the current path are terminated. Therefore these local updates are implemented as atomic actions and require the constant number of rounds.

As a result, the time complexity analysis under the above assumption is asymptotically equivalent to that analyzed assuming finer atomicity.

In this subsection, we use the time complexity of the previous algorithm *Zigzag*: $O(|P|)$, where $|P|$ is the length of the initial given path. Using this time complexity, we can present the following lemma.

Lemma 9. *If no vertical direction precedes any horizontal directions in P, our algorithm transforms P to a shortest path in $O(|P|)$ synchronous rounds.*

Proof. If there is no vertical direction in P, a shortest path can be constructed in $O(|P|)$ synchronous rounds, because the local update *V-Shift* is never executed in P and this implies that our algorithm uses only 3 rules which are the same as the algorithm *Zigzag*.

Even if there are some vertical directions in P, the local update *V-Shift* is never executed because the local update *V-Shift* is enabled when a vertical direction precedes a horizontal direction. In our proposed algorithm, no rule makes a vertical direction precede a horizontal direction. Thus, the path P can be divided into two disjoint sub-path (subsequence) P_p and P_v which consist of only horizontal directions and vertical directions respectively. P_p trivially precedes P_v and this order is

never changed. Note that P_p or P_v can be a empty sequence. Therefore, the time complexity of our algorithm under this assumption becomes the longer one between the convergence time (the number of synchronous rounds required) that takes to construct a shortest path from P_p and P_v. There is no local update enabled on the border of two sub-path P_p and P_v.

The time complexity of our algorithm in the case of P_p becomes $O(|P|)$ because there is no vertical direction in P_p.

Each router in P_v can detect the local update *ShortCut1* only, due to the conditions of the local update rules: *ShortCut2* and *Zigzag* is enabled by only horizontal directions, and *V-Shift* is enabled by a horizontal direction precedes a vertical direction. *ShortCut1* makes the path shorten by 2, hence, *ShortCut1* can be executed at maximum $\frac{|P_v|}{2}$ times. This implies the time complexity of our algorithm in the case of P_v becomes $O(|P|)$, because $|P_v| \leq |P|$.

As a result, our algorithm transforms a path P to a shortest path $O(|P|)$ if no vertical direction precedes any horizontal directions in the path P. □

Now we estimate the time complexity to transform any arbitrary path into the path in which no vertical direction precedes any horizontal directions.

In our algorithm, even if a router p_i detects a local update rule, this local update can not executed if $p_{(i-1)}$ detects a local update rule or $p_{(i-2)}$ detects either *ShortCut2* or *Zigzag* (refer line 12 to 17 in Algorithm 1). From the conditions of local update rule in our algorithm, the following lemma can be hold if p_i detects the local update rule *V-Shift*.

Lemma 10. *If a router p_i detects the local update rule V-Shift, this local update is never interrupted by the other local update rules except ShortCut1.*

Proof. From our algorithm, when a router p_i detects the local update rule *V-Shift*, $D_{(i-1)}$ is a vertical direction and D_i is a horizontal direction. The local update rules *ShortCut2* and *Zigzag* can be detected only when all $D_{(i-1)}$, D_i and $D_{(i+1)}$ are horizontal directions. This means that if p_i detects *V-Shift*, both $p_{(i-1)}$ and $p_{(i-2)}$ detect neither *ShortCut2* nor *Zigzag*. Moreover, $p_{(i-1)}$ never detects *V-Shift* because $D_{(i-1)}$ is a vertical direction.

As a result, *V-Shift* detected by p_i can be interrupted by *ShortCut1* detected by $p_{(i-1)}$. □

Using Lemma 10, we can also present the following lemma as follows.

Lemma 11. *If a router p_i detects the local update rule V-Shift, $D_{(i-1)}$ moves to the behind D_i or removed from the path in constant synchronous rounds.*

Proof. If a router p_i detects the local update rule *V-Shift*, p_i executes this local update unless $p_{(i-1)}$ detects *ShortCut1* by Lemma 10.

If $p_{(i-1)}$ detects *ShortCut1*, p_i executes local update in the next synchronous round (after $p_{(i-1)}$'s local update). Certainly $p_{(i-1)}$'s local update can be interrupted by the other local update rules detected by $p_{(i-2)}$ or $p_{(i-3)}$. However, in this case, $D_{(i-2)}$ a vertical direction, because p_i detects *V-Shift* and $p_{(i-1)}$ detects *ShortCut1*. This implies that $p_{(i-1)}$'s local update can be interrupted by only *ShortCut1* detected by $p_{(i-2)}$. As the same manner, $p_{(i-2)}$'s local update can be interrupted by only *ShortCut1* detected by $p_{(i-3)}$. As a result, the local update *V-Shift* can be only interrupted by the chain of preceding *ShortCut1*.

Since the length of the path is finite, the local update *ShortCut1* in the lead of the chaining *ShortCut1* (consists of vertical directions) can be executed repeatedly in each synchronous round. Therefore, *V-Shift* is executed or removed by the local update *ShortCut1* by $p_{(i-1)}$ after the finite number of the synchronous rounds because the length of the path is finite. Trivially, whether the execution (of *V-Shift*) or remove will be occurred depends on the number of the preceding *ShortCut1*. □

From Lemma 11, if a vertical direction immediately precedes a horizontal direction in the path, the vertical direction changes its position into the behind of the horizontal direction or removed from the path within the finite number of the synchronous rounds. In other words, a vertical direction immediately followed by a horizontal direction disappears after the finite number of synchronous rounds from the behind of the horizontal direction. Now we consider the number of rounds to make no vertical direction precede any horizontal directions.

Lemma 12. *Let P be an arbitrary initial given path, our proposed algorithm make no vertical direction precede any horizontal directions in P within $O(|P|)$ synchronous rounds.*

Proof. Let D_i be the last vertical direction which precedes any horizontal directions. $p_{(i+1)}$ detects *V-Shift* because D_i is a vertical direction and $D_{(i+1)}$ is a horizontal direction (line 9 and 10 in Algorithm 1).

From Lemma 11, D_i is removed by p_i's *ShortCut1* or moved to the behind of $D_{(i+1)}$ (a horizontal direction) by $p_{(i+1)}$'s *V-Shift* within the finite number of synchronous rounds. As a result, D_i is no longer the last vertical direction preceding any horizontal directions. We denote the number of synchronous rounds required to be moved or removed T_i: p_i can execute *ShortCut1* or $p_{(i+1)}$ can execute *V-Shift* after T_i synchronous rounds. This implies that D_i becomes no longer the last vertical direction preceding any horizontal directions after $(T_i + 1)$ synchronous rounds.

Because the length of the path is finite, the last vertical direction preceding any horizontal direction no longer exists by repeating $(T_i + 1)$ synchronous rounds. Assume that the execution of (T_i+1) synchronous rounds is repeated k times, to make no vertical direction precede any horizontal directions, $\sum_{i=1}^{k}(T_i + 1)$ synchronous rounds are required. Thus,

$$\begin{aligned}\sum_{i=1}^{k}(T_i + 1) &= k + (\sum_{i=1}^{k} T_i) \\ &< |P| + (\sum_{i=1}^{k} T_i) \quad (\because k < |P|) \\ &< |P| + \frac{|P|}{2} \quad (\because \sum_{i=1}^{k} T_i < \frac{|P|}{2}) \\ &= O(|P|)\end{aligned} \tag{1}$$

Note that every synchronous round of T_i makes P shorten by 2 (Lemma 11), hence the total number of T_i rounds becomes less than $\frac{|P|}{2}$. As a result of (1), our algorithm make no vertical direction precede any horizontal directions within $O(|P|)$ synchronous rounds. □

The following Theorem holds by Lemmas 9 and 12.

Theorem 2. *Our proposed algorithm transforms any given path P in a 3D grid network to a shortest path in $O(|P|)$ synchronous rounds.*

5.2 Experimental evaluation

Our proposed algorithm guarantees a construction of a shortest path from any initial given path in 3D virtual grid networks. And we analyze our algorithm in the previous subsection. In this subsection, we present our experimental results to show the practical performance of our proposed algorithm.

5.2.1 Overview of experiment

We implement the simulator which constructs a 2D/3D virtual grid network and executes our proposed algorithm. The simulator initially positions the source node and the destination node at the same coordination in the virtual grid network, and moves the destination node n times in a random direction. n is the length of the initial given path and it can be freely decided. In this simulator, the scale of the grid network is finite but large enough (*enough* means that the path does not arrive the end of the virtual grid network).

In this evaluation, we perform an experiment using our simulator while changing the length of the initial path, and we measure the number of synchronous rounds until the termination (no more local updates) of our algorithm. The goal of this evaluation is to clarify the actual number of the synchronous rounds which is required to construct a shortest path.

The time complexity of our algorithm is the same as the previous algorithm *Zigzag* for 2D virtual grid networks. Thus we compare the actual number of synchronous rounds between our proposed algorithm and the previous algorithm *Zigzag*.

5.2.2 The actual number of the synchronous rounds

In this experiment, we evaluate the number of synchronous rounds to construct a shortest path when an arbitrary path of length n is given. We count the number of synchronous rounds 10 times while changing the length of the path from 10 to 100 in increments of 10. And we repeat this experiment 1000 times on each length n and calculate the average number of the synchronous rounds to construct a shortest path.

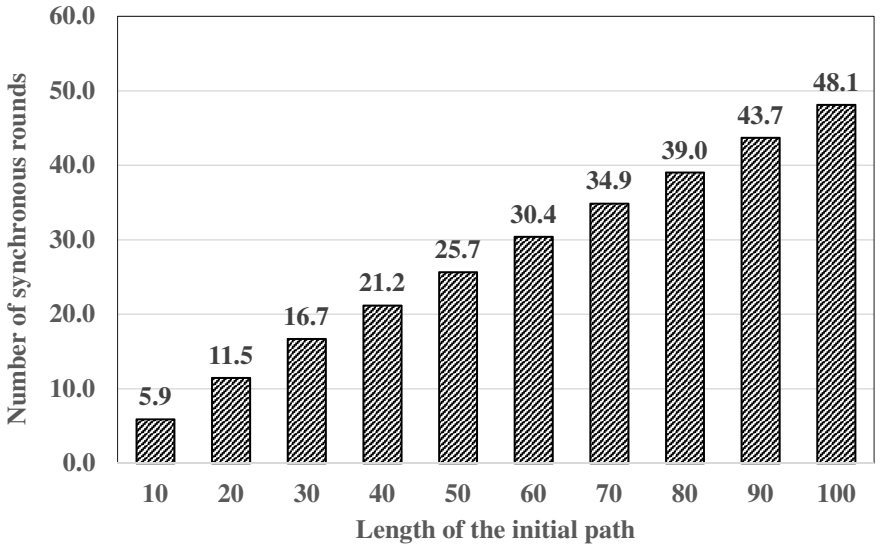


Figure 14: The number of synchronous rounds required to construct a shortest path

Figure 14 represents the average number of the synchronous rounds to construct a shortest path when an arbitrary path of length n is given. From this result, the average number of rounds increases linearly as the length of the path increases. This result is consistent with our analysis in the previous subsection. To easy to check that the average number of rounds is linear, we calculate the ratio of the number of synchronous rounds to the length of the initial path n like as Figure 15.

In Figure 15, although the ratio of the number of rounds slightly decreases according to the length n, it keeps a substantially constant ratio. If the length of the initial path becomes longer, the ratio of the synchronous rounds to the length n to construct a shortest path is slightly reduced because our algorithm can be executed concurrently at many routers which detect a local update rule in the path.

The average number of the synchronous rounds when the length of the initial path is long is presented in Table 1.

Length n	Average Rounds	Ratio
1000	409.53	40.95%
5000	1987.61	39.75%
10000	3949.31	39.49%

Table 1: The number of rounds when the length of the initial path is long

From table 1, the ratio of the synchronous rounds to the length n for constructing a shortest path becomes smaller when an initial path becomes longer.

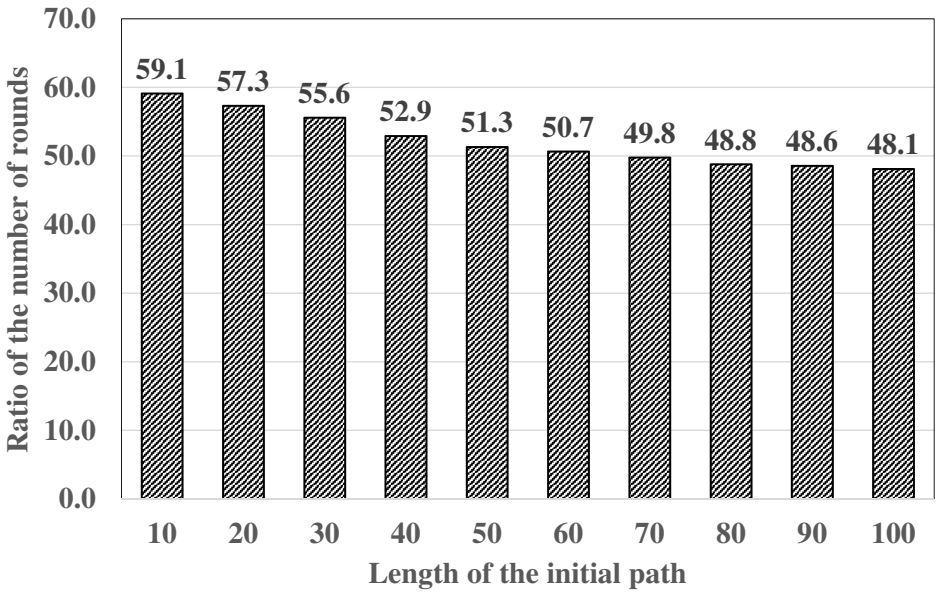


Figure 15: The ratio of the number of rounds to the length n

5.2.3 The comparison with the previous work

In the previous section, we had analyzed the time complexity our algorithm. For this analysis, we use the time complexity of the previous work: a self-optimizing routing algorithm *Zigzag* in a 2D grid network. In other words, the time complexity of our algorithm is based on the previous work. Nonetheless the time complexity of our algorithm has the orderly same ($O(|P|)$, where $|P|$ is the length of the initial path) as the one of the previous work, obviously our algorithm requires more synchronous rounds.

In this experiment, we verify the difference of the time complexity between our work and the previous work from a practical viewpoint[3].

Figure 16 represents the number of synchronous rounds of our algorithm and the previous work. In our algorithm, when the length of the path n is 10, a shortest path can be constructed in about 60% of n, and when n becomes longer by 100, the number of rounds decreases to about 50% of n. However, in the previous work *Zigzag*, it can construct a shortest path in a 2D grid network in about 30% of n when n is 10, furthermore, the number of rounds decreases to about 15% of n.

From this result, even though the assumptive system models are different, the previous work is superior to parallelism of the local updates, because the requiring number of rounds decreases more and more as the length of the path becomes longer.

In our algorithm, when many local update rules are detected at the same synchronous round and they are interrupted by the other local update rules in a continuous fashion, only one local update which is the most nearest one to the *source* node can be executed. On the other hand, the previous work *Zigzag* has some priorities among local update rules, thus two or more rules can be executed when many local update rules are collided sequentially. We consider that such a priority among local update rules can be adopted to our algorithm and it may decrease the actual number of synchronous rounds to construct a shortest path in 3D grid network.

[3]Actually our algorithm can be executed in both 2D and 3D virtual grid networks but the previous work operates only in 2D virtual grid networks, thus the result of this experiment is offered for reference.

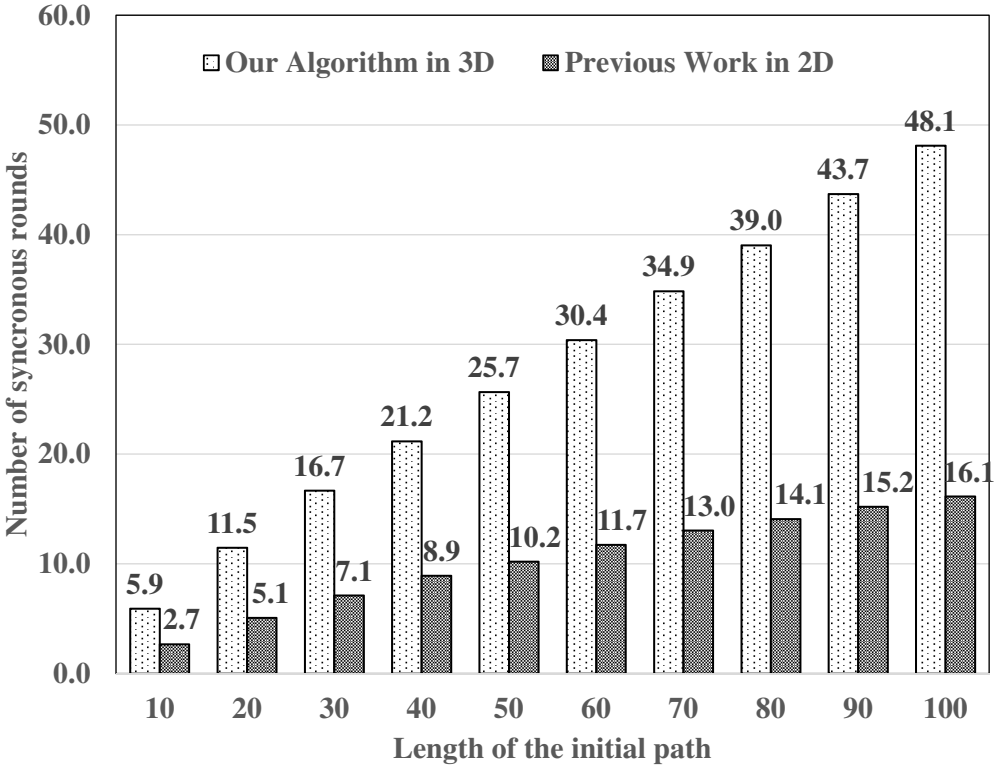


Figure 16: Comparison between our work in 3D grid and previous work in 2D grid

6 Conclusion

In this paper, we proposed a self-optimizing routing algorithm which construct the shortest path in a 3D virtual grid network. Our algorithm uses each router's local information only and has only four local update rules. We proved that our algorithm eventually constructs a shortest path from the *source* node to the *destination* node.

We had analyzed the complexity of our algorithm based on the previous work, and we presented the actual number of synchronous rounds to construct a shortest path using our algorithm.

To guarantee the correct local updates, we assume that each router maintains 6 hops of routing information, because some collisions can be occurred among local updates. We expect that this routing problem in a 3D virtual grid network can be resolved using less routing information (e.g. 4 hops for each router).

As we mentioned in the previous section, we expect that the actual number of synchronous rounds to transform an arbitrary given path to a shortest path can be decreased if we change the priority rules among local update rules in our algorithm[4].

Finally, to make our algorithm a practical protocol, the design of each local update as the distributed manner is required.

References

[1] S. Takatsu, F. Ooshita, H. Kakugawa, and T. Masuzawa, Zigzag: Local-information-based self-optimizing routing in virtual grid networks, Proceedings of the 33rd International Conference on Distributed Computing Systems (ICDCS) , pp. 358-368, July, 2013.

[4]The complexity of our algorithm may not be changed.

[2] Chai-Keong Toh, "Ad Hoc Mobile Wireless Networks: Protocols and Systems 1st Edition," Prentice Hall PTR, 2002.

[3] C. de Morais Cordeiro and D. P. Agrawal, "Ad Hoc and Sensor Networks: Theory and Applications (2nd Edition)," World Scientific, 2011.

[4] J. Zheng and A. Jamalipour, "Wireless Sensor Networks : A Networking Perspective," Wiley-IEEE Press, 2009.

[5] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01), pp.70–84, 2001.

[6] W. Dargie and C. Poellabauer, "Fundamentals of Wireless Sensor Networks: Theory and Practice," John Wiley & Sons, Ltd, 2010.

[7] C. Perkins and E. Royer, Ad hoc on demand distance vector routing, in Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 99), 1999, pp. 90100.

[8] J. N. Al-Karaki, A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," IEEE Wireless Communications, Vol. 11, Issue 6, pp.6–28, 2004.

[9] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, Energy-Efficient Communication Protocol for Wireless Microsensor Networks, Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS 00), 2000.

[10] D. Braginsky and D. Estrin, Rumor Routing Algorithm for Sensor Networks, in the Proceedings of the First Workshop on Sensor Networks and Applications (WSNA), pp.22–31, 2002.

[11] F. Ye, H. Luo, J. Cheng, S. Lu, L. Zhang, A Two-tier data dissemination model for large-scale wireless sensor networks, Proceedings of ACM/IEEE MOBICOM, pp.148–159, 2002.