symmetry

MDPI

Article

Recursively Divided Pancake Graphs with a Small Network Cost

Jung-Hyun Seo [1] and Hyeong-Ok Lee [2,*]

[1] Department of Multimedia Engineering, National University of Chonnam, Yeosu 59626, Korea; jhseo@scnu.ac.kr
[2] Department of Computer Education, National University of Sunchon, Sunchon 315, Korea
* Correspondence: oklee@scnu.ac.kr; Tel.: +82-61-750-3345

Abstract: Graphs are often used as models to solve problems in computer science, mathematics, and biology. A pancake sorting problem is modeled using a pancake graph whose classes include burnt pancake graphs, signed permutation graphs, and restricted pancake graphs. The network cost is degree × diameter. Finding a graph with a small network cost is like finding a good sorting algorithm. We propose a novel recursively divided pancake (RDP) graph that has a smaller network cost than other pancake-like graphs. In the pancake graph P_n, the number of nodes is $n!$, the degree is $n - 1$, and the network cost is $O(n^2)$. In an RDP_n, the number of nodes is $n!$, the degree is $2\log_2 n - 1$, and the network cost is $O(n(\log_2 n)^3)$. Because $O(n(\log_2 n)^3) < O(n^2)$, the RDP is superior to other pancake-like graphs. In this paper, we propose an RDP_n and analyze its basic topological properties. Second, we show that the RDP_n is recursive and symmetric. Third, a sorting algorithm is proposed, and the degree and diameter are derived. Finally, the network cost is compared between the RDP graph and other classes of pancake graphs.

Keywords: pancake graphs; network cost; $d \times k$; sorting; symmetric graph

Citation: Seo, J.-H.; Lee, H.-O. Recursively Divided Pancake Graphs with a Small Network Cost. *Symmetry* **2021**, *13*, 844. https://doi.org/10.3390/sym13050844

Academic Editor: Serge Lawrencenko

Received: 16 April 2021
Accepted: 7 May 2021
Published: 10 May 2021

1. Introduction

The pancake graphs originated with pancake sorting problems. An n-pancake sorting problem refers to the task of sorting n pancakes of varied sizes piled on a stack. The stacked states are represented as permutations of the set {1, …, n}. The operation of reversing the positions of the pancakes from the top to the k-th pancake is called prefix reversal (*flip* hereinafter). A *flip*, therefore, changes the positions of the first k ($1 < k \leq n$) pancakes. When sorting is finished, symbol 1 is positioned at the top, and symbol n at the bottom [1]. The solution to this problem is as follows. First, put the largest pancake on the top, and then take it down to the bottom, then repeat this iteration with the rest of the pancakes $n - 2$ times. Simply put, the number of flips in order to complete the sort in the worst time complexity is $2n - 3$. W. Gates suggests a method in which the number of required flips is $\fallingdotseq 1.6666n$ and the lower bound is proved to be $\fallingdotseq 1.0625n$ ($n \equiv 0 \pmod{16}$) [2], while B. Chitturi suggests a method in which the number of flips is $\fallingdotseq 1.6363n$ and the lower bound is proved to be $\fallingdotseq 1.071n$ ($n \equiv 0 \pmod{16}$) [3].

The n-pancake sorting problem can be represented by the pancake graph P_n. The permutations are mapped to the nodes and *flips* to the edges. The number of nodes in P_n is n, and the degree is $n - 1$. The operation that *flips* the kth pancake is $flip_k$. In graph theory, the *degree* of a node is the number of edges incident with the node. The degree of a graph represents the maximum degree of all nodes. The distance between two nodes, u and v, is the number of edges of the shortest path between u and v. The diameter is the maximum distance between all pairs of nodes [4].

Time complexity is a measure used to evaluate sorting algorithms. It refers to the number of times the main operations are performed so that the algorithm can terminate.

There are three main operations for sorting the permutations, as is the case for the classes of star or pancake graphs: *reversals* (*flips*) that flip the order of symbols, *transpositions* that move the symbols to other positions, and *translocations* that exchange two symbols with each other.

Direct comparison of sorting algorithms using these three operations is difficult, because the given main operations all differ. Suppose we are given an arbitrary node, $S = s_1s_2s_3 \ldots s_{n-2}s_{n-1}s_n$. Then, the main operation of a star graph exchanges s_1 and s_k $(1 < k \leq n)$. However, the main operation of a pancake graph is the *flip*. The sorting time complexity of a star graph S_n is $\fallingdotseq 1.5n$, and that of a pancake graph P_n is $\fallingdotseq 1.67n$ [5]. Thus, it is not reasonable to simply compare the time complexities of these algorithms. In an extreme case, if there is a graph wherein all symbols can move to all positions, the time complexity is $n - 1$. However, it cannot be said that this sorting algorithm is effective. When evaluating time complexity, the number of main operations must also be evaluated.

When a sorting problem is expressed as a graph, the number of main operations is mapped to the degree and the worst time complexity to the diameter. The network cost is degree × diameter. The network cost is the sorting cost of the sorting algorithm. If an edge is added to the graph, the degree increases, and the diameter decreases. Conversely, if an edge is removed, the degree decreases and the diameter increases. These two factors provide a trade-off. Therefore, it is difficult to reduce network costs [6]. The degree of a pancake graph is $n - 1$, and the network cost is $\boldsymbol{O}(n^2)$. We propose an RDP that reduces the degree to $\boldsymbol{O}(\log_2 n)$ and the network cost to $\boldsymbol{O}(\log_2 n^3)$ by using recursively divided edges.

Pancake-like graphs have been studied for sorting problems [2,3,7–10], interconnection networks in computer science [11–16], and DNA computing models in biology [17–21]. Finding a graph with a small network cost is like a high-performance interconnection network and a good sorting algorithm [22]. The burnt pancake problem involves the sorting of pancakes that are burnt on one side in order of their size. W. Gates proved that the lower bound for the number of flips is $1.5n$ and the upper bound is $2n + 3$ in the sorting of n burnt pancakes [2]. D. Cohen proved that the lower bound for the number of flips ($n \geq 10$) is $1.5n$ and the upper bound is $2n - 2$ [23]. V. Bafna introduced research on the sorting permutations to the genes of animals and plants in order to translate between two permutations [24]. P. Berman proposed an algorithm that completes a sort $\fallingdotseq 1.375n$ number of flips [17]. A signed permutation by reversal (SBR) is a problem that involves the sorting of permutations that consist of an integer between $-n$ and n, excluding 0. S. Hannenhalli proposed an algorithm in which the upper bound of the number of flips is $O(n^2)$ [18]. The result of the research in this paragraph is introduced in [6].

In this paper, we propose an RDP graph that has a lower network cost than pancake-like graphs. In Section 2, the notations used throughout the paper are described, and the RDP is defined. In Section 3, we analyze the properties of the RDP. In Section 4, a sorting algorithm is introduced, and diameters and network costs are analyzed and compared. Finally, Section 5 presents the conclusions.

2. Preliminaries and Definition of Recursively Divided Pancake

This section provides the definitions used in this paper, and a recursively divided pancake RDP_n is described.

An RDP divides the symbols in half and performs a *flip* operation using edges. These edges are generated recursively. In $flip_k$, the pancake graph is $1 < k \leq n$. With an RDP, $k = n, n/2 + 1, n/2, n/4 + 1, n/4, n/8 + 1, n/8, \ldots, 3, 2$. Thus, k divides n recursively.

For a node address of RDP_n, a permutation of natural numbers from 1 to n is used. Let an arbitrary node, $S = s_1s_2s_3 \ldots s_{k-1}s_ks_{k+1} \ldots s_{n-2}s_{n-1}s_n$, exist. Definition 2 defines the *flip* operation and, in Definition 3, symbol s_k at the kth position is moved to the top. The symbols that should be carefully considered throughout this paper are written in blue.

Definition 1. s_k *is the kth symbol or position k symbol.* $|S|$ *denotes the number of symbols.*

Definition 2. The flip operation for S is $FO_k(S) = s_k s_{k-1} s_{k-2} \ldots s_3 s_2 s_1 s_{k+1} \ldots s_{n-2} s_{n-1} s_n$, $k = n, n/2 + 1, n/2, n/4 + 1, n/4, n/8 + 1, n/8, \ldots, 3, 2$.

Definition 3. The moving operation for s_k to the *top* is $MTF_k(S) =$

MTF$_k$*(S) {*
  if ($|S| == 1$) then return
  if ($|S|/2 < k$) then {
    FO_n
    $k = (n - k) + 1$
  }
  $MTF_k(s_1 s_2 s_3 \ldots s_{n./2})$
}

The definition of a recursively divided pancake RDP_n is as follows:

$RDP_n = (V, E) \mid n = 2^m$, where m is the natural number.

node $V = \{s_1 s_2 s_3 \ldots s_{k-1} s_k s_{k+1} \ldots s_{n-2} s_{n-1} s_n \mid 1 \le s_k \le n, 1 \le k \le n\}$.

The edges of the RDP_n are defined as follows:

flip edge $FE_k = (S, FO_k(S))$.

Figure 1 shows RDP_4. The solid lines forming a hexagon are FE_2 and FE_3, and the blue line connecting a hexagon and another hexagon is FE_4.

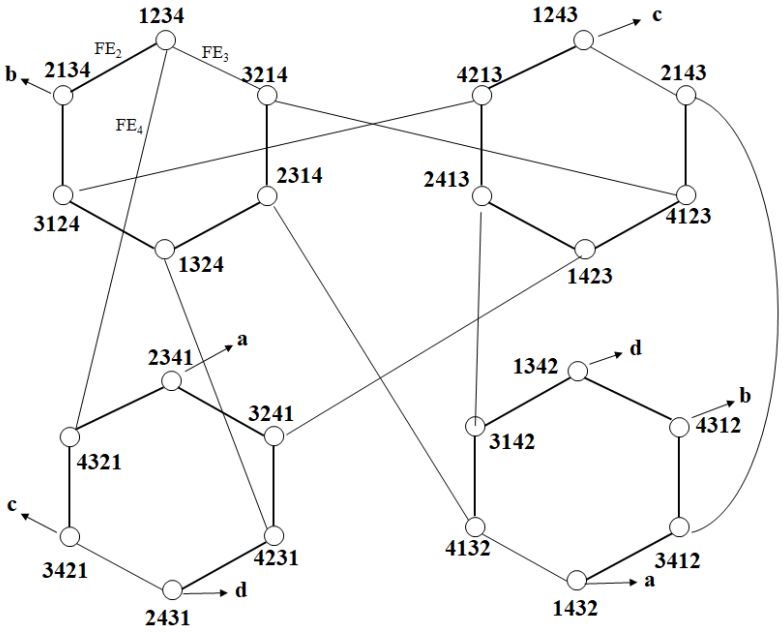

Figure 1. Four-dimensional recursively divided pancake RDP_4.

For example, in RDP_8, the (connecting edge, neighboring node) pairs for node 12345678 are as follows:

(FE_2, 21345678), (FE_3, 32145678), (FE_4, 43215678), (FE_5, 54321678), (FE_8, 87654321).

A path can be expressed as a node sequence from the start node to the destination node. Here, a neighboring node is connected by an edge. The paths are represented as follows:

Notation 1. → *is used to indicate a path. For example, the paths between 1234, 2134, 3124, and 4213 in Figure 1 can be represented as follows:*

1234 → 2134 → 3124 → 4213 or

1234 → FE_2 → 2134 → FE_3 → 3124 → FE_4 → 4213.

3. Topological Properties

In this section, the basic topological properties and the recursive expandability of the RDP are examined. Then, we prove that it is a connected graph.

Theorem 1. *RDP_n is a regular graph with a degree of $2\log_2 n - 1$, and the number of nodes is $n!$. The number of edges is $n!\log_2 n - 0.5n$.*

Proof. Because the node addresses of RDP_n are permutations of natural numbers from 1 to n, the total number of nodes is $n!$. The number of edges FE connected to an arbitrary node is $2\log_2 n - 1$. Because every node has the same number of edges, RDP_n is regular, and the degree is $2\log_2 n - 1$. Because it is an undirected graph, the total number of edges is $(n! \times (2\log_2 n - 1))/2 = n!\log_2 n - 0.5n!$. □

For a graph to be expanded recursively, two conditions must be satisfied. First, the expansion rules must be consistent. Second, the graph of level n must contain a graph of level $n - 1$. Recursive graphs are favorable for sorting algorithms, broadcasting algorithms, and algorithms for path problems, because the algorithm at the $n - 1$ level is used the same way as in an algorithm at level n.

Theorem 2. *RDP_n expands recursively.*

Proof. First, nodes are examined, after which the edges are explained. Here, $|RDP_n|$ is the number of nodes in RDP_n. In RDP_n, the number of nodes increases as the level increases based on the following method: $|RDP_2| = 2$. $|RDP_4| = |RDP_2| \times 3 \times 4$. $|RDP_8| = |RDP_4| \times 5 \times 6 \times 7 \times 8$. $|RDP_n| = |RDP_{n/2}| \times (n/2) + 1 \times (n/2) + 2 \times (n/2) + 3 \times \cdots \times n$.

Therefore, $|RDP_n|$ is defined recursively as follows:

$$|RDP_n| = |RDP_{n/2}| \times (n/2) + 1 \times (n/2) + 2 \times (n/2) + 3 \times \cdots \times n.$$

Let us now examine the edges. Figure 2 provides arrows to indicate the positions that can be flipped in the node addresses for RDP_2, RDP_4, RDP_8, and RDP_{16} from left to right. See the red arrows. Even if the level increases, the flip position of the previous level remains intact. In other words, RDP_n maintains the edges of RDP_{n-1} as they are. See the green arrows. As the level increases, edges are added at positions $n/2 + 1$ and n recursively. Therefore, RDP_n expands recursively. □

(**a**) RDP_2 (**b**) RDP_4 (**c**) RDP_8 (**d**) RDP_{16}

Figure 2. *flip* position.

If all nodes are connected in a graph, it is a connected graph that has a path from an arbitrary node to every other node. A node address is comprised of permutations. If a symbol that forms a permutation can be moved to any position, then all permutations are connected to each other. If a symbol, s_1, can be moved to any other position, those of all positions can also be moved to s_1. This ensures that all symbols can be moved to any position.

Theorem 3. *RDP_n is a connected graph.*

Proof. RDP_4 and RDP_8 are examined sequentially, and the graph is generalized for RDP_n. Suppose $S = s_1s_2s_3s_4$ in RDP_4. s_1 can be moved to any position.
$FO_2(S) = s_2s_1s_3s_4$, $FO_3(S) = s_3s_2s_1s_4$, $FO_4(S) = s_4s_3s_2s_1$.

Therefore, RDP_4 is a connected graph. Suppose $S = s_1s_2s_3s_4s_5s_6s_7s_8$ in RDP_8. It was shown in RDP_4 that s_1 can move to positions 2, 3, and 4. Here, if FO_8 is added, s_1 moves to positions 5, 6, 7, and 8.

$$s_8s_7s_6s_5s_1s_2s_3s_4 = FO_8(FO_4(S)).$$

$$s_8s_7s_6s_5s_4s_1s_2s_3 = FO_8(FO_3(S)).$$

$$s_8s_7s_6s_5s_4s_3s_1s_2 = FO_8(FO_2(S)).$$

$$s_8s_7s_6s_5s_4s_3s_2s_1 = FO_8(S).$$

Therefore, RDP_8 is a connected graph. According to these results, s_1 can move to positions 2, 3, 4, ⋯, $n/2$ in RDP_n. Here, if FO_n is added, s_1 moves to positions $n/2 + 1$, $n/2 + 2$, ⋯, n. Therefore, RDP_n is a connected graph. □

It is well known that a permutation (e.g., a star graph) is a symmetry group [11]. Symmetry groups are automorphic. Because the nodes of RDP_n are permutations, the proof of node transitivity is omitted in the proof of Theorem 4.

Theorem 4. *RDP_n is a symmetric graph.*

Proof. Only the edge transitivity is proved. Let an arbitrary node be $s = s_1s_2s_3 \dots s_k \dots s_{n-1}s_n$. The mapping function is $f = s_1s_2s_3 \dots s_k \dots s_{n-1}s_n \rightarrow s_1s_2s_3 \dots s_k \dots s_ns_{n-1}$. The mapping function, f, is a bijection, which can be any function that exchanges symbols. The edge, FE_k, is denoted by (s, t). It is shown that $f(s)$ and $f(t)$ are connected to each other.

If $s = s_1s_2s_3 \dots s_{k-1}s_ks_{k+1} \dots s_{n-2}s_{n-1}s_n$, then $t = s_ks_{k-1} \dots s_3s_2s_1s_{k+1} \dots s_{n-2}s_{n-1}s_n$. $f(s) = s_1s_2s_3 \dots s_{k-1}s_ks_{k+1} \dots s_{n-2}s_ns_{n-1}$, and $f(t) = s_ks_{k-1} \dots s_3s_2s_1s_{k+1} \dots s_{n-2}s_ns_{n-1}$. Thus, $f(s)$ and $f(t)$ are connected to FE_k. Therefore, RDP_n is a node- and edge-transitive graph and is symmetric. □

4. Sorting Algorithm and Network Cost Comparison

In this section, we first present a sorting algorithm. Subsequently, the diameter is derived to calculate the network cost. Then, the network cost is compared to that of other pancake-like graphs.

Let us assume that a start node is u, and a destination node is v. A sorting algorithm is a rule that generates a sequence of nodes from u to v. In other words, it changes the address of the start node to that of the destination node by using the *flip* operator, FO_k. If the destination node $v = 1234 \dots n$, then it is a sorting algorithm.

A rough overview of sorting is as follows. First, the symbols of u are divided into two bundles, uf and ur, as shown in Figure 3. Then, symbols 5 and 6 in uf are exchanged with symbols 1 and 2 in ur. When this step is completed, uf and vf comprise the same symbols. This is also true for ur and vr. If the above process is repeated in uf and ur until the number of symbols becomes 1, u will be the same as v.

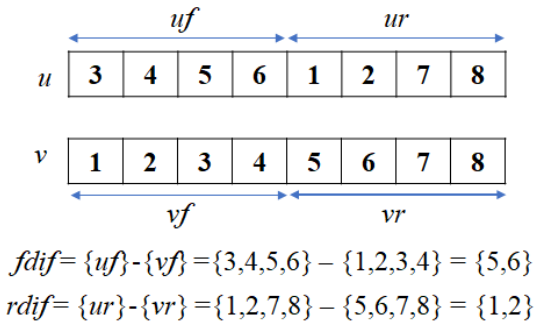

Figure 3. <*Pre-work*> in RDP_8.

In the sorting algorithm, <*pre-work*> defines several factors. In particular, it defines the symbols that should be exchanged between *uf* and *ur*. In Figure 3, 56 and 12 are symbols that should be exchanged. Let the start node be $u = u_1u_2u_3 \ldots u_{k-1}u_ku_{k+1} \ldots u_{n-2}u_{n-1}u_n$, and the destination node be $v = v_1v_2v_3 \ldots v_{k-1}v_kv_{k+1} \ldots v_{n-2}v_{n-1}v_n$. <*pre-work*>.

In the address of node *u*, a set $uf = \{u_1,u_2,u_3, \ldots, u_{n/2}\}$ and a set $ur = \{u_{n/2+1}, \ldots u_n\}$. In the case of node *v*, $vf = \{v_1,v_2,v_3, \ldots, v_{n/2}\}$ *and* $vr = \{v_{n/2+1}, \ldots v_n\}$.

Set *fdif* = {*uf*} − {*vf*} and set *rdif* = {*ur*} − {*vr*}. There is a sequential order in *fdif* and *rdif*, and the *m*-th *rdif* element is written as *rdif*[*m*].

In Figure 3, for example, if *u* = 34561278 and *v* = 12345678, then *uf* = {3,4,5,6}, *vf* = {1,2,3,4}, *ur* = {1,2,7,8}, and *vr* = {5,6,7,8}. Here, *fdif* = {5,6}, *rdif* = {1,2}, |*fdif*| = |*rdif*| = 2, *fdif*[1] = 5, and *fdif*[2] = 6.

The sorting algorithm in Algorithm 1 is performed recursively until |*uf*| = |*ur*| = 1. The sorting algorithm is as follows: = = is an equality operator, and = is an assignment operator.

Algorithm 1. sorting(u) {

```
1:   if (|u| ==1) return;
2:   <pre-work >
3:   if (|fdif| > n/4) {
4:      u = u → FO_{n/2};
5:      <pre-work >
6:    }
7:   exchange(u);
8:   sorting(uf, vf) ;
9:   u = u → FO_{n/2};
10:  sorting(uf, vr);
11:  u = u → FO_{n/2};
12: }
```

The function *exchange*() in Algorithm 2 exchanges *fdif* in *uf* with *rdif* in *ur*. If *exchange*() is executed once, then there is no additional symbol exchange with *uf* and *ur*.

Algorithm 2. exchange(u) {

```
1:  for (m = 1; m ≤ |fdif| ; m + +) {
2:     k = fdif[m];
3:     MTF_k(uf);
4:     FO_{n/2};
5:     FO_n;
6:     k = rdif[m] ;
7:     MTF_k(uf) ;
8:     FO_{(n/2)+1};
9:  }
```

10: **if** (|*fdif*| is odd) FO_n;
11: }

The Algorithm 2 exchanges symbols based on the following method.
Move the first symbol to be exchanged to u_1 via MTF().
Flip a half through $FO_{n/2}$.
Flip the whole thing through FO_n.
Move the second symbol to be exchanged to u_1 via MTF().
Exchange u_1 and $u_{(n/2)+1}$ with each other through FO_{n+1}.

The diameter is the distance between the two farthest nodes in the graph. The diameter is an important measure for evaluating a graph, and, in the worst case, it is the sorting time complexity. It represents the software cost when the graph is implemented in a system.

Theorem 5. *The diameter of* RDP_n *is* $0.5n(\log_2 n)^2 + n\log_2 n$.

Proof. The diameter analysis is substituted by finding the worst-case time complexity of the sorting algorithm. The number of calls, a, of *sorting*(u) is called recursively as follows: $|u|$ and $|v|$ are reduced by half each time the function is called, and the function is terminated when the size becomes one. Therefore, $a = \log_2 n$. In the worst case, the number of symbols exchanged is $b = n/2$ when the function is called once in the sorting algorithm. The operations required to exchange two symbols with each other are MTF() and FO, which are performed twice and thrice, respectively. If the number of exchanged symbols is odd, FO is required only once. The number of times MTF() is executed for a symbol is $\log_2 n$ in the worst case. The number of operations required to exchange the two symbols is $c = 2\log_2 n + 3 + 1$.

The diameter, $k = a \times (bc)/2$, where a is the number of times that *sorting*() is called, and $(bc)/2$ is the internal time complexity of *sorting*().

$$k = a \cdot (bc)/2.$$
$$= \log_2 n \cdot ((n/2 \times (2\log_2 n + 4))/2).$$
$$= \log_2 n \cdot n/4(2\log_2 n + 4).$$
$$= \log_2 n(0.5n\log_2 n + n).$$
$$= 0.5n(\log_2 n)^2 + n\log_2 n$$

. □

Network cost is an important measure for evaluating graphs. Table 1 shows the network cost of RDP_n vs. various pancake-like graphs. The diameter and degree of pancake [2,10], restricted pancake [6], burnt pancake [1,2], restricted burnt pancake [6], and signed permutation [8,10] have been presented in a previous study [6]. The network cost is shown by the Big-O notation.

Table 1. Comparison of network cost with other classes of pancake graphs.

Graph.		No. of Node	Degree	Diameter	Network Cost
pancake	[2]	$n!$	$n-1$	$\boldsymbol{O}(5n/3)$	$\fallingdotseq 1.67n^2$
	[9]			$\boldsymbol{O}(18n/11)$	$\fallingdotseq 1.64n^2$
restricted pancake	[6]	$n!$	$n/2+1$	$\boldsymbol{O}(3.5n)$	$\fallingdotseq 1.7n^2$
burnt pancake	[2]	$2^n \times n!$	$n(n+1)/2$	$\boldsymbol{O}(3n/2)$	$\fallingdotseq 0.75n^3$
	[1]			$\boldsymbol{O}(23n/14)$	$\fallingdotseq 0.82n^3$
restricted burnt pancake	[6]	$2^n \times n!$	$n/2+2$	$\boldsymbol{O}(4.5n)$	$\fallingdotseq 2.25n^2$
signed permutation	[8]	$2^n \times n!$	$n(n-1)/2$	$\boldsymbol{O}(n^2)$	$\fallingdotseq 0.5n^4$
	[10]			$\boldsymbol{O}(n^{1.5})$	$\fallingdotseq 0.5n^{3.5}$
RDP		$n!$	$2\log_2 n - 1$	$\boldsymbol{O}(n(\log_2 n)^2)$	$\fallingdotseq 2n(\log_2 n)^3$

Because $\boldsymbol{O}((\log_2 n)^3) < \boldsymbol{O}(n^2)$, RDP is superior to other pancake-like graphs as shown in Table 1 in terms of network cost.

5. Conclusions

Pancake-like graphs are used in various fields wherein network cost is an important evaluation measure. We have provided an RDP that has a smaller network cost than existing pancake-like graphs. In RDP_n, the number of nodes was $n!$, the degree was $2\log_2 n - 1$, and the diameter was $0.5n(\log_2 n)^2 + n\log_2 n$. The network cost of RDP_n was $\boldsymbol{O}((\log_2 n)^3)$. we propose an RDP that has a lower network cost than pancake-like graphs. The pancake-like graphs have been studied for sorting problems, interconnection networks, and DNA computing models. We hope that RDP will be used as a model to solve problems in various fields. We analyzed the topological properties of the RDP and connected graphs and proposed a more efficient sorting algorithm. It is expected that various algorithms, such as broadcasting, parallel path, spanning tree, and embedding algorithms, can be improved based on this graph in the future. Time complexity is a measure used to evaluate sorting algorithms. It refers to the number of times the main operations are performed so that the algorithm can terminate. I wonder if it is correct to compare sorting algorithms using only time complexity. I think the number of main operations can also be included in the evaluation. I think this idea needs more discussion.

Author Contributions: Conceptualization, J.-H.S. and H.-O.L.; methodology, J.-H.S.; software, J.-H.S.; validation, J.-H.S.; formal analysis, J.-H.S.; investigation, J.-H.S.; resources, J.-H.S.; data curation, J.-H.S.; writing—original draft preparation, J.-H.S.; writing—review and editing, J.-H.S.; visualization, J.-H.S.; supervision, H.-O.L.; project administration, H.-O.L.; funding acquisition, H.-O.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2020R1A2C1012363).

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Jiang, H. Target Set Selection on generalized pancake graphs. *Indian J. Pure Appl. Math.* **2020**, *51*, 379–389, doi:10.1007/s13226-020-0406-8.
2. Gates, W.H.; Papadimitriou, C.H. Bounds for sorting by prefix reversal. *Discret. Math.* **1979**, *27*, 47–57, doi:10.1016/0012-365x(79)90068-2.
3. Yamamura, A.; Csuhaj-Varjú, E.; Dömösi, P.; Vaszil, G. Rearrangement Problem of Multidimensional Arrays by Prefix Reversals. In Proceedings of the 15th International Conference on Automata and Formal Languages, Debrecen, Hungary, 4–6 September 2017; pp, 9–10.
4. Cheng, D.-W.; Chan, C.-T.; Hsieh, S.-Y. Constructing Independent Spanning Trees on Pancake Networks. *IEEE Access* **2019**, *8*, 3427–3433, doi:10.1109/access.2019.2962549.
5. Mendia, V.; Sarkar, D. Optimal broadcasting on the star graph. *IEEE Trans. Parallel Distrib. Syst.* **1992**, *3*, 389–396, doi:10.1109/71.149958.
6. Seo, J.-H.; Kim, J.-S.; Lee, H.-O. An algorithm for sorting pancake by restricted reversals. *J. Supercomput.* **2015**, *71*, 3832–3850, doi:10.1007/s11227-015-1473-1.
7. Araki, T.; Horiyama, T.; Nakano, S.I.; Okamoto, Y.; Otachi, Y.; Uehara, R.; Uno, T.; Yamanaka, K. Sorting by Five Prefix Reversals. *IPSJ SIG Tech. Rep.* **2020**, *2020-AL-179*, 1–7.
8. Blanco, S.A.; Buehrle, C.; Patidar, A. On the number of pancake stacks requiring four flips to be sorted. *arXiv* **2019**, arXiv:1902.04055.
9. Chitturi, B.; Fahle, W.; Meng, Z.; Morales, L.; Shields, C.O.; Sudborough, I.H.; Voit, W. An (18/11)n upper bound for sorting by prefix reversals. *Theor. Comput. Sci.* **2009**, *410*, 3372–3390.
10. Tannier, E.; Bergeron, A.; Sagot, M.-F. Advances on sorting by reversals. *Discret. Appl. Math.* **2007**, *155*, 881–888, doi:10.1016/j.dam.2005.02.033.
11. Akers, S.; Krishnamurthy, B. A group-theoretic model for symmetric interconnection networks. *IEEE Trans. Comput.* **1989**, *38*, 555–566, doi:10.1109/12.21148.

12. Wang, R.; Zhu. Q. The h-extra conditional diagnosability of burnt pancake networks under the PMC model. In Proceedings of the 2017 First International Conference on Electronics Instrumentation & Information Systems (EIIS), Harbin, China, 3–5 June 2017; pp. 1–6.
13. Heydari, M.H.; Sudborough, I. On the Diameter of the Pancake Network. *J. Algorithms* **1997**, *25*, 67–94, doi:10.1006/jagm.1997.0874.
14. Yang, Y.-C.; Kao, S.-S.; Klasing, R.; Hsieh, S.-Y.; Chou, H.-H.; Chang, J.-M. The Construction of Multiple Independent Spanning Trees on Burnt Pancake Networks. *IEEE Access* **2021**, *9*, 16679–16691, doi:10.1109/access.2021.3049290.
15. Yeh, C.-H.; Varvarigos, E. Macro-star networks: Efficient low-degree alternatives to star graphs for large-scale parallel architectures. In Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96); Institute of Electrical and Electronics Engineers (IEEE), Annapolis, MD, USA, 27–31 October 2002; pp. 290–297.
16. Pai, K.-J.; Chang, R.-S.; Chang, J.-M. Constructing dual-CISTs of pancake graphs and performance assessment of protection routings on some Cayley networks. *J. Supercomput.* **2021**, *77*, 990–1014, doi:10.1007/s11227-020-03297-9.
17. Berman, P.; Hannenhalli, S.; Karpinski, M. 1.375-Approximation Algorithm for Sorting by Reversals. In Proceedings of the 10th European Symposium on Algorithms, Rome, Italy, 17–21 September 2002; pp. 200–210.
18. Elias, I.; Hartman, T. A 1.375-Approximation Algorithm for Sorting by Transpositions. In *International Workshop on Algorithms in Bioinformatics*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 204–215.
19. Grusea, S.; Labarre, A. The distribution of cycles in breakpoint graphs of signed permutations. *Discret. Appl. Math.* **2013**, *161*, 1448–1466, doi:10.1016/j.dam.2013.02.002.
20. Haynes, A.K.; Broderick, M.L.; Brown, A.D.; Butner, T.L.; O Dickson, J.; Harden, W.L.; Heard, L.H.; Jessen, E.L.; Malloy, K.J.; Ogden, B.J.; et al. Engineering bacteria to solve the Burnt Pancake Problem. *J. Biol. Eng.* **2008**, *2*, 1–12, doi:10.1186/1754-1611-2-8.
21. Heyer, L.J.; Poet, J.L.; Broderick, M.L.; Compeau, P.E.C.; Dickson, J.O.; Harden, W.L. Bacterial Computing: Using E. coli to Solve the Burnt Pancake Problem. *Math Horiz.* **2010**, *17*, 5–10, doi:10.4169/194762110x489242.
22. Seo, J.; Lee, H.O. Design and analysis of the rotational binary graph as an alternative to hypercube and Torus. *J. Supercomput.* **2020**, *76*, 7161–7176.
23. Cohen, D.S.; Blum, M. On the problem of sorting burnt pancakes. *Discret. Appl. Math.* **1995**, *61*, 105–120, doi:10.1016/0166-218x(94)00009-3.
24. Bafna, V.; Pevzner, P.A. Genome rearrangements and sorting by reversals. *SIAM J. Comput.* **2002**, *25*, 148–157.