湖南大学学报（自然科学版）
Journal of Hunan University (Natural Sciences)

第 50 卷 第 12 期
2023 年 12 月

Available online at http://jonuns.com/index.php/journal/index

Vol. 50 No. 12
December 2023

Open Access Article

https://doi.org/10.55463/issn.1674-2974.50.12.14

Hybrid Approach for Discovering *k*-Hamiltonian Paths in a Torus-Enhanced Butterfly Interconnected Network

Latifah[1]*, Tubagus Mohammad Akhriza[2]

[1] School of Informatics Management and Computer, JAKARTA STI&K, South Jakarta, Indonesia

[2] Pradnya Paramita School of Informatics Management and Computer, Malang, East Java, Indonesia

* Corresponding author: latifahbahrudinsuryobroto@gmail.com

Received: September 16, 2023 / Revised: October 10, 2023 / Accepted: November 13, 2023 / Published: December 29, 2023

Abstract: The significance of interconnection networks extends beyond the semiconductor industry, becoming integral to industrial engineering, particularly in the era of the Internet of Things (IoT). Both sectors share a strategic emphasis on developing sophisticated networks to enhance performance and scalability. Techniques such as Cartesian product fusion have led to the development of novel interconnected networks, addressing the demand for high-speed communication. The novel network topology, including the Torus-Enhanced Butterfly (TREB) network, is developed by the Cartesian product of the Torus (TR) and Enhanced Butterfly (EB) networks and holds unexplored aspects, particularly concerning the existence of Hamiltonian paths. The objective of this study is to devise a methodology for identifying k-Hamiltonian paths in an extensive TREB network. Contributions include theoretical and algorithmic proofs of Hamiltonian path existence using a hybrid approach. This approach divides the large TREB network graph into four EB subgraphs, and from each subgraph, all Hamiltonian paths are sought using the brute force method as a subsolution. Next, a dynamic programming-based algorithm is used to connect the Hamiltonian paths in these four subgraphs into a complete Hamiltonian path in the TREB network. The experiments reveal several properties of the TREB network that can only be derived through algorithmic approaches. These include the exact number of paths, which can be found from a valid permutation arrangement of the EB subgraphs, reaching approximately one billion paths, and the sequence of nodes forming Hamiltonian pathways.

Keywords: Hamiltonian paths, torus network, butterfly network, interconnection network, exact algorithms.

环形增强蝶形互联网络中 k-哈密尔顿路径的混合方法

摘要：互连网络的重要性超越了半导体行业，成为工业工程不可或缺的一部分，特别是在物联网(物联网)时代。这两个部门的战略重点都是开发复杂的网络以提高性能和可扩展性。笛卡尔积融合等技术促进了新型互连网络的发展，满足了高速通信的需求。这种新颖的网络拓扑，包括环面增强型蝴蝶(TREB)网络，是由环面(TR)和增强型蝴蝶(EB)网络的笛卡尔积开发的，并且具有未探索的方面，特别是关于哈密顿路径的存在。本研究的目的是设计一种方法来识别广泛的 TREB 网络中的 k-哈密尔顿路径。贡献包括使用混合方法对哈密顿路径存在性进行理论和算法证明。该方法将大型 TREB 网络图划分为 4 个 EB 子图，并使用蛮力法作为子解从每个子图中寻找所有哈密顿路径。接下来，使用基于动态规划的算法将这四个子图中的哈密顿路径连接成 TREB 网络中的完整哈密顿路径。实验揭示了 TREB 网络的几个属性，

这些属性只能通过算法方法得出。其中包括路径的确切数量（可以从 EB 子图的有效排列排列中找到，达到大约十亿条路径）以及形成哈密顿路径的节点序列。

关键词：汉密尔顿路径, 环面网络, 蝶形网络, 互连网络, 精确算法.

1. Introduction

The interconnection network plays a pivotal role in the Industry 4.0 framework, which is characterized by the seamless integration of advanced technologies in manufacturing and industrial processes. This framework significantly intersects with the burgeoning concept of the Internet of Things (IoT), in which an interconnected network serves as the backbone that unites a myriad of objects. These objects encompass a diverse range, including wireless sensors, electrical components, electronic devices, mechanical systems, and cutting-edge 5G technology. Within the expansive landscape of IoT, the interconnected network serves as the linchpin, facilitating seamless communication and collaboration among these varied elements. This network extends its reach beyond the physical realm to encompass virtual data and environments, thereby establishing a comprehensive connectivity framework facilitated by the internet [1]-[2]. Within this evolving landscape, the present study conducts a comparative analysis of diverse interconnection networks, each representing a unique approach to achieving optimal communication and processing speeds [3]-[5].

The development of interconnection networks is also integral to the dynamic landscape of the semiconductor industry, which is characterized by a continual increase in the number of components integrated onto a chip. This upward trajectory has engendered a heightened demand for efficient core interconnections, which has catalyzed the adoption of Network on Chip (NoC) as a promising solution. In stark contrast to conventional on-chip interconnections, the NoC demonstrates superior effectiveness in facilitating communication among nodes. A paramount consideration in this context revolves around the intricate design of routing algorithms, which are pivotal for the seamless transmission of packets between communicating nodes within the NoC. Current research endeavors are concentrated on systematically addressing the challenges inherent in routing algorithms, categorizing them based on diverse criteria such as aging awareness, thermal awareness, congestion awareness, fault awareness, resilience, and energy efficiency. The design of a NoC routing algorithm capable of providing less congested paths, heightened energy efficiency, and superior scalability constitutes a formidable challenge [6]-[7].

Various techniques have been employed to augment network capabilities and ensure adaptability to the evolving demands of industrial processes. One notable technique involves the fusion of different network topologies through methods such as Cartesian product, tensor product, or lexicographic product [5], [8]-[9]. By combining the strengths of distinct topologies, engineers aim to create novel network structures that inherit desirable properties from each component. For instance, the application of tensor or lexicographic products has led to the development of interconnected networks such as the Star-Cube, Hyper-Butterfly, Torus-Embedded Hypercube, and Scalable Twisted Hypercube [8]-[14].

Similarly, the application of the Cartesian product has been instrumental in crafting pioneering topologies, as exemplified by the Torus-Enhanced Butterfly (TREB) network. This configuration is the result of combining a Torus (TR) network with dimensions of 2 2 nodes and an Enhanced Butterfly (EB) network with dimensions of 3 23 nodes. Notably, this composite structure reveals the attachment of four EB subgraphs to each node in the TR network, with inter-subgraph paths following the routes delineated in the TR [31, 32]. These references have delved into the structural properties, embedding characteristics, and enumeration aspects of the expansive TREB network topology. However, the crucial matter of the existence of Hamiltonian paths within this novel topology has yet to be thoroughly explored, thus becoming the central focus of attention in this article.

A Hamiltonian path in a network visits each node exactly once. In other words, it is a sequence of nodes in which every node is visited exactly once, and every edge of the network is traversed exactly once. If the Hamiltonian path forms a cycle by connecting the starting and ending nodes, it is referred to as a Hamiltonian cycle. The significance of Hamiltonian paths in interconnection networks, such as those in industrial engineering and the semiconductor industry, lies in their ability to establish efficient and comprehensive communication routes [15]-[18]. However, it is also crucial to note that the algorithmic search for Hamiltonian paths poses a challenge as it falls under the category of NP problems. The problem signifying that finding a Hamiltonian path is a computationally complex problem where verifying a potential solution can be done quickly, but finding the solution itself may require an impractical amount of time [15]-[17].

The objective of this study is to propose a

methodology to mathematically prove the existence of Hamiltonian paths in the TREB network graph and to uncover algorithmic properties that can be effectively obtained through algorithmic proofs. These properties include the exact number of paths and the sequence of nodes that form Hamiltonian pathways.

The contributions of this study are threefold. First, in addition to theoretically proving the existence of Hamiltonian paths, it also demonstrates their existence algorithmically through a hybrid approach. Second, the approach first divides the graph of the extensive TREB network topology into four EB subgraphs positioned at the nodes of the Torus graph. All Hamiltonian paths in each subgraph are initially found using the brute force method and are stored. In the subsequent steps employing the dynamic programing approach, the four Hamiltonian paths within each subgraph are linked using bridging edges, thereby establishing comprehensive k-Hamiltonian paths within the TREB network. This process exhibits significant speed in identifying millions to billions of paths, as opposed to the impracticality of searching for all paths using the brute-force method within a given time tolerance. Additionally, visualization of the discovered paths is capable of illustrating how paths in these subgraphs are interconnected in one visit.

This article subsequently presents the results of a literature review of Torus and enhanced butterfly networks, Torus-enhanced butterfly networks, and several approaches to searching for Hamiltonian paths. This is followed by a section discussing the proposed solution method and experimental design. In the next section, the experimental results and their discussion are provided. The article concludes with a summary and plans for future development.

Computer interconnection networks have been widely applied in various areas, such as parallel computing systems, multi-processor systems, and workstation networks [19]-[20]. The network topology (model) of the interconnection is important for parallel processing or distributed systems. For this reason, a topology with high connectivity is preferred. As stated in [21]-[22], a good interconnection network topology must have symmetrical properties, be scalable, and have a small diameter. Scalable means that network nodes must have a constant degree, whereas a good topology has constant and limited degrees. Furthermore, connectivity is widely used to measure network capacity, whereas diameter indicates routing efficiency (data transmission). A good network topology also needs to have a Hamiltonian path, which is needed when a message must be sent sequentially from one processor to another in the network exactly once [15], [23]-[24].

2. Literature Review

2.1. Torus and Enhanced Butterfly Network

Suppose G is a network graph, denoted as G = (V, E, T), where vertices V are the network nodes, edge E is the link or channel between nodes, and T is the traffic on the link [25]. In this study, to show the existence of a Hamiltonian path in a network graph, traffic T is replaced with path P; thus, the graph notation becomes G = (V, E, P). Supposed v_1 and v_2 are nodes in G, then E(v_1, v_2) denotes an edge or link between v_1 and v_2, while P shows the path direction between two v_1 and v_2, thus it has two possible values: (v_1, v_2) and (v_2, v_1) $\in$ E that show there exists path from v_1 to v_2 and v_2 to v_1 respectively. The direction of this path is important to define because in a network, two nodes can have more than one link to ensure the delivery of data packets to all nodes in the network.

Torus networks are the development of mesh networks with dimensions represented by the number of processor nodes in the rows and columns of the network. A 4 4-node mesh network is shown in Fig. 1 (left), while a 4 4-node Torus Network is shown on the right.

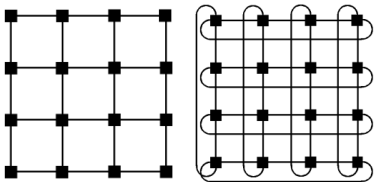

Fig. 1 Mesh (left) and torus network graph (right) [4], [26]

As shown in Fig. 1, a torus network is described as a network that connects the heads of columns, the tail of the column, and the left side of the row to the right-side row. Torus networks have better path diversity than mesh networks, and they also have more minimal routes. Several publications discuss the application of mesh and Torus networks in designing chip networks [4], [26], and in interconnection networks [3], [27]-[29].

According to [30], a wrap-around butterfly network with degree n ≥ 3, B_n, has $n2^n$ nodes and $n2^{n+1}$ edges. An enhanced butterfly network, EB of degree 3, EB_3 as in Fig. 3, has the same number of nodes as B3, namely 3 x 8 = 24 nodes, but the number of edges is added so that the degree becomes 5. The number of edges becomes: degree × $n2^{n-1}$. This is made clear by Fig. 2, which shows that the degree of EB3 is 5, with the number of edges = $5 \times 3 \times 2^2$ = 60 edges.

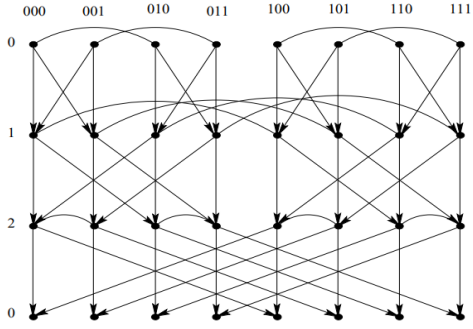

Fig. 2 Enhanced butterfly network graph 3 x 8 nodes [30]

2.2. Torus-Enhanced Butterfly Network

Recent studies proposed developing an EB_3 network into a *TR*(2, 2) network through a Cartesian product, thus producing a *TREB*(2, 2, 3) network [31]-[32]. According to these publications, some properties related to this new network are explained as follows.

Definition 1: A cycle that contains all vertices of the graph is called a Hamiltonian cycle [33].

Definition 2: A graph G with a Hamiltonian cycle is said to be a Hamiltonian graph [33].

Theorem 1: The Cayley graph has a Hamiltonian cycle [34].

Proposition 1: Every graph that has a Hamiltonian cycle will have a Hamiltonian path [12].

Theorem 3: Torus graph is a Cayley graph [35].

Theorem 4: The enhanced butterfly graph is a Cayley graph [30].

Proposition 1: The Cartesian product of two Cayley graphs is also a Cayley graph [32].

The following lemmas are proposed in [31]-[32]:

Lemma 1: Torus-Enhanced Butterfly Interconnection Network topology is a Cayley graph.

Proof: Torus and enhanced butterfly interconnection networks are Cayley graphs. By proposition 1, the Cartesian product between the two interconnection networks Cayley graphs is also a Cayley graph; thus, the Torus-Enhanced Butterfly interconnection network is a Cayley graph.

Definition 3: If the TR network graph has dimension of (m, l) and EB_n has dimension of $(n, 2^n)$, then the TREB network, denoted as $TREB(m, l, n)$, is the Cartesian product of Torus and Enhanced Butterfly. This is true for $n \geq 3$, $m \geq 2$ and $l \geq 2$.

Lemma 2: TREB (m, l, n) network graph has a constant node degree of nine.

Proof: since Torus has degree 4 and EB_3 has degree 5, thus, *TREB*(2, 2, 3) has 4 + 5 = 9 degree.

Lemma 3: $TREB(2, 2, 3)$ consists of 96 nodes.

Proof: Since Torus has 4 nodes and EB_3 has 24 nodes, thus, Cartesian product of the two networks produces 4 x 24 = 96 nodes.

Lemma 4: $TREB(2, 2, 3)$ consists of 432 edges.

Proof: Since the *TREB* graph has a degree of nine, the number of edges = degree $\times (m \times l) \times n2^{n-1}$ edges $= 9 \times 2 \times 2 \times 3 \times 2^2 = 432$ edges.

2.3. Methods for Finding Hamiltonian Paths in Networks

In the literature, the exploration of Hamiltonian paths in networks involves various methods, including exact algorithms, approximate algorithms, and heuristic algorithms. These methodologies identify Hamiltonian paths efficiently although each approach has distinct characteristics and trade-offs [36]-[37].

Exact algorithms rigorously determine Hamiltonian paths by exhaustively exploring all possible solutions. These algorithms systematically evaluate every potential Hamiltonian path, ensuring a precise and accurate solution. While providing guaranteed accuracy, exact algorithms may face computational challenges, especially with large networks, due to the exhaustive nature of the search. Brute force, dynamic programming, and branch and bound are three examples of exact algorithms used for solving combinatorial optimization problems such as finding Hamiltonian paths.

Brute force is a straightforward approach in which the algorithm systematically generates all possible combinations or permutations of paths in the network. This method explores the entire solution space exhaustively, ensuring that no potential solution is missed. While brute force guarantees an optimal solution, it can be computationally expensive, especially for large networks, because of the shear number of possibilities. Dynamic programing breaks down a complex problem into smaller overlapping subproblems, solving each subproblem only once and storing the solutions for future reference. By avoiding redundant calculations, dynamic programing reduces the overall computational effort. It provides an exact solution with improved efficiency compared to brute force. However, its application might be constrained by the nature of the problem and the available memory. The branch and bound approach involves systematically exploring paths, pruning branches, and continuing the search until an optimal solution is found [38]-[45].

Approximate algorithms seek Hamiltonian paths with a focus on speed, allowing faster computation at the expense of potentially suboptimal solutions. These algorithms provide quick solutions that are close to the optimal Hamiltonian path but do not guarantee an exact match. Approximate algorithms are valuable for large-scale networks where achieving an exact solution within a reasonable timeframe may be impractical. This algorithm includes the nearest neighbor, genetic algorithm, and ant colony optimization.

Nearest Neighbor starts from a specific node and selects the nearest neighbor at each step, aiming to find a path quickly. This is an approximate approach because it does not guarantee an optimal solution but provides a solution that is usually close to optimal. Genetic algorithms mimic the process of natural selection by evolving a population of potential solutions over multiple iterations. Genetic algorithms are approximate, providing a good solution but not necessarily the best one. Ant Colony Optimization is inspired by the foraging behavior of ants, using pheromone trails to find and reinforce good paths. It is an approximate method that finds near-optimal solutions [37], [43], [46].

Heuristic algorithms prioritize speed and efficiency by employing rules of thumb or strategies that guide

the search for Hamiltonian paths. These algorithms leverage smart decision-making processes to explore the solution space more efficiently, often sacrificing optimality for speed. Heuristic algorithms are particularly useful in real-time or resource-constrained scenarios because they provide reasonably good solutions in a shorter timeframe. Included in this category are tabu search, simulated annealing, and multi-start local search.

Tabu Search is a heuristic method used to solve optimization problems. It is inspired by the way humans solve problems by learning from previous experiences. The algorithm maintains a set of “tabu” moves, which are moves that should be avoided in the short term to prevent cycling. Tabu Search explores the solution space by iteratively making moves, evaluating their quality, and updating the current solution based on certain criteria. It uses memory structures to avoid revisiting solutions and escape local optima. Simulated annealing is a probabilistic optimization algorithm inspired by the annealing process in metallurgy. It starts with an initial solution and iteratively explores the solution space by allowing "bad" moves initially, with decreasing probability over time. The algorithm accepts worse solutions to escape local optima but gradually reduces the probability of accepting worse solutions as it progresses. This allows the algorithm to explore a broader solution space. Multi-start local search is a metaheuristic that involves running a local search algorithm multiple times from different initial solutions. Each run explores a local neighborhood to find the local optimum. The algorithm then selects the best solution among all runs as the final output. This approach helps escape local optima by starting the search from various points in the solution space [47]-[53].

3. Proposed Methods

3.1. Proving the Hamiltonian Path in the TREB Network

In this study, the existence of a Hamiltonian path in a TREB graph is mathematically proven through the following lemma.

Lemma 5: The TREB(m, l, n) has a Hamiltonian path. Proof: because the topology of the TREB network is a Cayley graph and each Cayley graph has a Hamiltonian path; then, the topology of the TREB network has a Hamiltonian path.

However, despite mathematical evidence of the existence of Hamiltonian paths in TREB networks, practical implementation poses significant challenges because of the NP-complete nature of the problem.

One exact method for finding Hamiltonian paths is brute force, which explores all possible paths to discover Hamiltonian paths. The number of possible Hamiltonian paths in a complete graph with N nodes is expressed as N! Therefore, the complexity of searching for Hamiltonian paths using the brute-force method in a graph with N nodes is O(N!), making the task computationally demanding because of the factorial exponential growth.

This study employs a hybrid-based method, namely brute force and dynamic programing, to discover k Hamiltonian paths in a large TREB graph. Both of these exact methods are proposed in this study to demonstrate the accurate existence of Hamiltonian paths in the TREB graph. However, before explaining these two methods, the formation of the TREB graph is discussed first.

3.2. Generating a Large TREB Graph

Referring to the shape of the *TR*(2, 2) [4], [26], and EB_3 [30] graphs, the steps for forming the Torus-Enhanced butterfly are carried out in three steps: defining the Torus directed graph (digraph), defining the EB_3 digraph, and building the TREB digraph from the defined Torus and EB digraphs. A more detailed explanation is as follows.

1. Create the TR(2, 2) digraph:

i. TR = Digraph(2, 2) # prepare a (2, 2)-nodes digraph;

ii. Add nodes with labels ‘00’, ‘01’, ‘10’, ‘11’ to TR;

iii. Create two-way links between two nodes, as illustrated in Table 1.

Table 1 Torus network graph data (Developed by the authors)

v_1	v_2	P
00	10	$v_1 \rightarrow v_2, v_2 \rightarrow v_1$
10	11	$v_1 \rightarrow v_2, v_2 \rightarrow v_1$
11	01	$v_1 \rightarrow v_2, v_2 \rightarrow v_1$
01	00	$v_1 \rightarrow v_2, v_2 \rightarrow v_1$

A Torus *TR*(2, 2) digraph is formed as in Fig. 3.

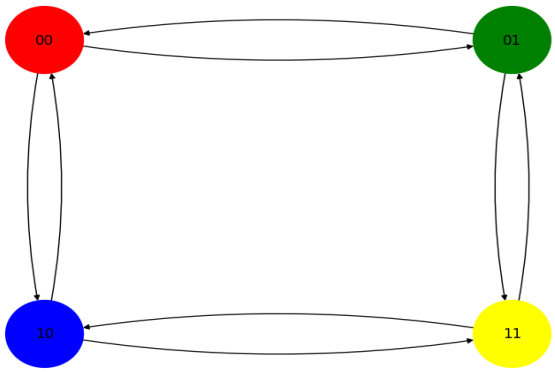

Fig. 3 The TR(2, 2) network graph (Developed by the authors)

2. Create the EB_3 digraph:

i. Create an EB_3 digraph:

ii. EB = Digraph(3, 8) # Prepare a (3, 8)-node digraph.

iii. Add nodes with labels from '0000' to '0111' in the first row, from '1000' to '1111' in the second row, and from '2000' to '2111' in the third row.

iv. Establish links between nodes and form an enhanced butterfly digraph, as illustrated in Fig. 4. Note that all links are unidirectional paths.

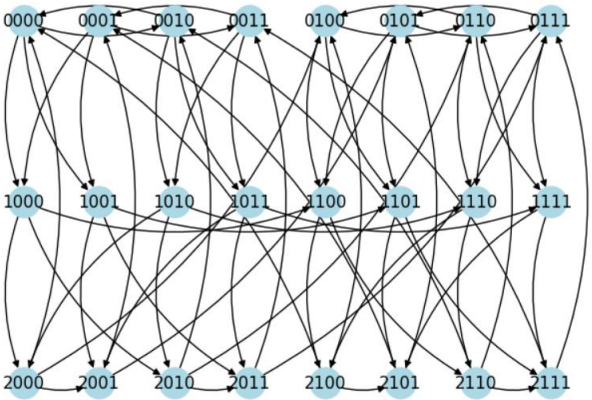

Fig. 4 The EB_3 network graph (Developed by the authors)

The EB_3 graph in Fig. 4 is identical to the graph in Fig. 2, except that the path from the nodes in the third row is directed back to the nodes in the first row. Additionally, four undirected edges, such as from 0000 to 0010, are programmatically represented as two directed edges going back and forth between these two points.

3. Create a TREB large graph:

i. TREB = TR × EB, thus the TREB digraph has dimensions of (2, 2, 3, 2^3), or is simplified as (2, 2, 3). Here, a new TREB graph with 432 edges and 96 nodes is created by combining nodes from TR and EB.

ii. Each node label is represented as '00'.'0001' (written as 000001), indicating that node 0001 in EB is embedded in node 00 in TR. This labeling approach is applied to all EB nodes embedded in other TR '10', '01', and '11' nodes.

The node labels in Torus are referred to as the prefixes of the nodes in TREB. For example, the node '000001' has the prefix '00,' indicating that this node is located at the position of the '00' node in Torus. The resulting TREB graph is visualized in Fig. 5, illustrating that EB_3 subgraph is installed at each node position of the TR.

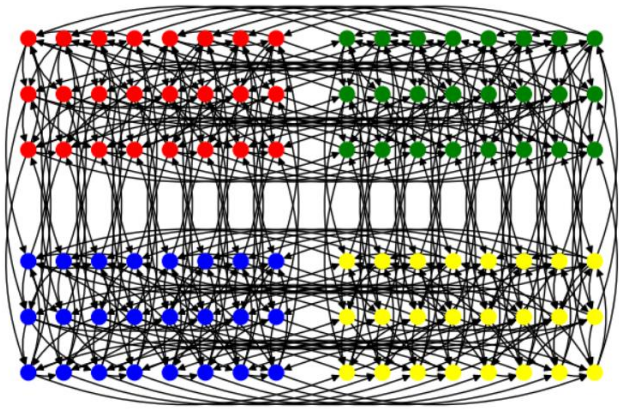

Fig. 5 Torus-enhanced butterfly network graph (Developed by the authors)

3.3. Finding All Hamiltonian Paths in the EB Subgraph

Before explaining the process of finding Hamilton paths in a large TREB network, we first explain the basic algorithm used in this study to find a Hamilton path in EB Subgraph G using a brute force approach. Here are the general steps or principles of the brute force method for finding the Hamiltonian path [37], [54]:

1. *Initialization:* Start with an empty path and choose a starting node.
2. *Expansion:* Extend the path by moving to an adjacent node that has not been visited; repeat this process until all nodes are included in the path.
3. *Validation:* Check if the final path forms a Hamiltonian path, i.e., it visits each node exactly once. If the last node in the path is connected to the starting node, it forms a Hamiltonian cycle:
4. *Backtracking:* If the current path is not a Hamiltonian path, backtrack to the previous decision point and explore alternative paths; continue this process until all possibilities have been explored.
5. *Completion:* Once a Hamiltonian path is found, the search process stops.
6. *Optimizations:* Depending on the specific implementation, optimizations may be applied to reduce the search space and improve efficiency. This could involve pruning branches that cannot lead to a valid Hamiltonian path.

Referring to these studies, three functions were developed to search for Hamiltonian paths using the brute force method, employing various functions and methods in the NetworkX library for the Python language. These include the G.*nodes* and G.*edges* properties, which return all the nodes and edges in G, respectively. The function G.*neighbors*(*S*) retrieves the neighboring nodes of node *S*, and G.*has_edge*(v_1, v_2) returns true if there is an edge between v_1 and v_2 in G.

These three functions are *nodes_traversal*, which searches node by node in G and forms a path. The *hamiltonian_check* function determines whether the path yielded from the first function is a Hamiltonian path or not. The third function, *Hamiltonian_search*, combines the two functions above to find all Hamiltonian paths in the given graph G. A more detailed explanation of the three functions is provided as follows.

Function nodes_traversal(G, P, S**)**

S is the starting node of the path being searched
P is a list that stores the path, which is a series of nodes

1. P = P + [S] # at first, insert the start node at the path's first index
2. if len(P) == len(G.nodes): yield *P*
3. For *nb* in G.neighbors(S):

```
        If nb is not in P:
        for new_P in nodes_traversal(G, nb, P): yield new_P
```

Function hamiltonian_check(G, P**)**:

```
1. if len(P) == len(G.nodes()):
2.         if not G.has_edge(P[-1], P[0]):
               return True
           else:
               return False #P is a cycle
3. elif len(P) == len(G.nodes()) - 1:
           return True
   else:
           return False #P is incomplete
```

Function HamiltonianPath_search(G**):**

```
1. paths = []
2. for S in G.nodes:
3.     for P in list(node_traversal(G, S)):
4.         if hamiltonian_check(G, P):
5.             paths.append(path)
6. return paths
```

The nodes_traversal(G, P, S) function explores all possible paths in the graph using a recursive depth-first search. Each node traverses its neighbors and continues the search until a Hamiltonian path is found or all possibilities are exhausted. The time complexity is O(N!), as it explores all possible permutations of nodes. The hamiltonian_check(G, P) function checks whether a given path P is a Hamiltonian path or not. It verifies whether the path length is equal to the number of nodes and whether the last node is connected to the starting node. The time complexity is O(1) because the checks involve constant time operations. The HamiltonianPath_search(G) function calls *nodes_traversal* to generate paths and then checks each path using hamiltonian_check. The overall time complexity depends on the number of paths generated and checked. In the worst case, if all possible paths are explored, the complexity is O(N!). In practice, the actual complexity may be less than O(N!) depending on the structure of the graph and the early termination when a Hamiltonian path is found.

3.4. Discovering k-Hamiltonian Paths in the TREB Network

The proposed method finds *k-Hamiltonian* paths in the TREB network using a dynamic programing approach. As explained earlier, the TREB network graph is actually formed by four EB subgraphs installed at four nodes in TR. After all Hamilton paths in the EB subgraphs are found, the next step is to connect these Hamilton paths in the four subgraphs to form a complete Hamiltonian path for the TREB network. The method of connecting these Hamilton paths can be considered a dynamic programing approach because the algorithm breaks down the problem into smaller subproblems (building a series of paths) and systematically solves them. In addition, it uses the results of smaller subproblems to build more complex solutions (the complete Hamiltonian path in TREB).

The proposed k-Hamiltonian paths search method is excluded from Branch and Bound because it often involves pruning the search space based on certain criteria. The proposed algorithm, as described above, does not use such pruning or bounding strategies. Instead, it focuses on systematically building paths and series of paths until the desired number (*k*) is achieved.

The proposed *k*-Hamiltonian path finding procedure is performed as follows:

Procedure Find_kpaths

Initialization

1. Find all Hamilton paths in each subgraph Sub00, Sub01, Sub10, and Sub11, and save the paths found in each subgraph, respectively in the lists H00, H01, H10, and H11. Here the HamiltonianPath_Search(G) function is used, where G is replaced by each subgraph mentioned.

Note: Here, the label xx in Subxx and Hxx is the prefix of nodes in the TREB indicating their positions in the Torus nodes. For example, Sub00 is the EB subgraph at the '00' Torus node position, simultaneously indicating that all nodes within it have the prefix '00'.

2. Discover bridges, which are edges (v_j, v_k) where v_j is the last node of the path $[v_1, v_2, ..., v_n = v_j]$ belonging to Hxx1, and v_k is the first node of the path $[v_1 = v_k, v_2, ..., v_n]$ belonging to Hxx2, where Hxx1 and Hxx2 are adjacent. Save these bridges in Bxx, where xx is the prefix of Hxx1. Due to the presence of four Hxx, there are only three Bxx located between each pair of adjacent Hxx.

3. The task of finding a Hamiltonian path in a large TREB graph is to form a series of Hamiltonian paths $\text{path0}.\text{b}_1.\text{path1}.\ \text{b}_2.\text{path2}.\ \text{b}_3.\text{path3}$ where path0, path1, path2, and path3 $\in$ H (= H00 $\cup$ H01$\cup$ H11 $\cup$ H10), while b_1, b_2 and $b_3 \in$ B (= B00 $\cup$ B01 $\cup$ B11) are the edge bridges that connect the paths between Hxx.

4. Set H, a list of Hxx sequences such that two Hxxs in sequence are adjacent, just like nodes in a Torus topology. For example, H00, H01, H11, and H10 are the correct sequences, while one of the sequences that will not find any path is H00, H11, H10, and H01 because H00 and H11 are not adjacent in Torus (see Fig. 3).

5. H = [H00, H01, H11, H10] # this is used only as a running example.

6. B = [B00, B01, B11] #bridges between adjacent Hxx in H.

7. Set k as the desired number of paths to be found.

Part #1: building a series of $\text{path0}.\text{b}_1.\text{path1}$ where path0$\in$ H[0], $b_1 \in$ B[0] and path1$\in$ H[1]

8. PART1 = [] #list of found paths

9. For path0 in H[0]:
10. P1 = [] # an empty list to store a path
11. Get the next bridge b1 = an edge $(v_j,v_h) \in$ B[0]
12. For path1 in H[1]:
13. If b1 = (path0[-1], path1[0]):
14. P1.append(path0, b1 and path1)
15. PART1.append(P1) # store the series
16. If len(PART1) >= *k*: break all loops

Part #2: continuing to build a series of path0, b1, path1, b2, and path2

17. PART2 = [] # list to store the found paths
18. Part1 = iter(PART1) # set PART1 as an iterable object
19. p0, b1, path1 = next(Part1) # get the first element in Part1
20. P2 = [] # an empty list to store a path
21. Get the next bridge b2 = an edge $(v_j,v_h) \in$ B[1]
22. For path2 in H[2]:
23. If b2 = (path1[-1], path2[0]):
24. P2.append(p0, b1, path1, b2, path2)
25. PART2.append(P2) # store the series
26. if len(PART2) >= *k*: break all loops
27. else: repeat from line 17 # get the next element in Part1

Part #3: continuing to build a series of path0, b1, path1, b2, path2, b3, and path3

28. PART3 = [] # list to store the found paths
29. Part2 = iter(PART2) # set PART2 as an iterable object
30. p0, b1, p1, b2, path2 = next(Part2) # get the first element in Part2
31. P3 = [] # an empty list to store a path
32. Get the next bridge b3 = an edge $(v_j,v_h) \in$ B[2]
33. For path3 in H[3]:
34. If b3 = (path2[-1], path3[0]):
35. P3.append(p0, b1, p1, b2, path2, b3, path3)
36. PART3.append(P3) # store the series
37. If len(PART3) >= *k*: break all loops
38. else: repeat from line 27 # get the next element in Part2

Lines 18 and 19 employ Python's *itertool* module, specifically using the *iter* and *next* functions, which play distinct roles when interacting with iterable objects. The *iter* function is employed to generate an iterator from an *iterable*, as exemplified by using *iter*(PART1) to make the PART1 list iterable. Conversely, the *next* function retrieves the subsequent element from an iterator. When used in tandem, *iter* and *next* offer a versatile and memory-efficient approach to accessing elements from an iterable one at a time. Iterators prove especially valuable when sequentially processing elements, allowing for efficient processing without the need to load the entire iterable into memory simultaneously. PART3 is a list of Hamiltonian paths found in the TREB graph.In Part #2, path 1 is extracted from Part1, and bridge b2 is taken from B, which connects the last element of path1 to the first element of path2. However, in practice, it is not guaranteed that a sequence p1, b2, path2 can be formed in a loop (lines 22 to 24) as many as *k* times. If the condition for k is not met, path 1 will be retrieved from Part1. As explained earlier, for efficiency reasons, elements in Part1 are not placed in a for loop altogether, but the *iter* and *next* functions are employed to retrieve elements one by one for the construction of the path sequence.

The aforementioned treatment is applied similarly to Part #3, and in its implementation, there is an indicator in a Part indicating where the path formation process ceases, enabling the iteration to continue within this Part. The objective is to ensure that from all starting paths of an EB subgraph, a complete Hamiltonian path can be formed in the larger TREB graph.

Instead of searching for all possible paths, this effort is relaxed to simply walking the paths that have been found in the EB subgraphs with a valid walking path, namely a path that follows the direction in the defined Torus graph. This activity is more guaranteed to find paths in a large TREB graph because we will only walk through the paths that are already available.

4. Experimental Work, Results, and Discussion

4.1. Experimental Settings

The purpose of this experiment is to test the capability of the brute-force-based program for discovering Hamiltonian paths in the EB subgraph and the dynamic programming-based program for finding k-Hamiltonian paths in the large TREB graph. All proposed functions and procedures are implemented in Python and executed on a laptop running Windows 11 64-bit, equipped with an Intel® Core™ i7-10750H CPU @ 2.60GHz, 2.59 GHz, and 8 GB RAM.

An additional observation is made regarding the *Find_kPaths* procedure in Part #2, as it may not consistently connect the paths formed in the previous two Hxxs to the path in the third Hxx within a single iteration. Consequently, iterations in Part #2 resume from the next path1 (inside Part1 using the next function on line 17). In Part #3, to discover as many Hamiltonian paths as possible, the maximum iteration limit is set equal to the number of paths in the fourth Hxx.

4.2. Hamiltonian Paths in the EB Subgraph

In the initialization stage, four Subxxs were formed, each containing 60 edges connecting 24 nodes in each Subxx. Additionally, three Bxxs were formed, each having 64 edges. These 64 bridges connect Subxx with two adjacent Subxx, each having 32 bridges. For

example, Sub00 is adjacent to both Sub01 and Sub10, so there are 32 bridges connecting Sub00 to Sub10 and from Sub00 to Sub01. In summary, the total number of edges in the four Subxxs is 4 x 60 = 240, plus the number of bridge edges in the three Hxxs = 3 x 64 = 192, resulting in 432 edges, thereby confirming its compliance with Lemma 4 regarding the number of edges in the TREB graph.

Executing three functions on each EB subgraph with 24 nodes and 60 edges yields 36,470 Hamiltonian paths, completing in an average time of 3.42 seconds, which is considered fast. However, divergent outcomes are observed when the program is executed for the TREB network graph, as it fails to generate output within the specified time tolerance of five minutes, leading to the termination of the program.

As a solution, a dynamic programing approach is applied to connect the already discovered Hamiltonian paths in each EB subgraph into a complete Hamiltonian path in the TREB network graph. The results are explained in the following section.

4.3. k-Hamiltonian Paths in the TREB Network

The attempt to discover k-Hamiltonian paths using various *k* values, ranging from k = 100 to three million (3000K), and the time (in seconds) required to discover these k-paths is elucidated in Fig. 6. This diagram indicates that the search time for paths in the extensive TREB graph exhibits a linear increase with the number of paths sought. Discovering 1000K paths takes only 26 seconds, 2000K requires approximately 56 seconds, while 3000K takes merely 81 seconds.

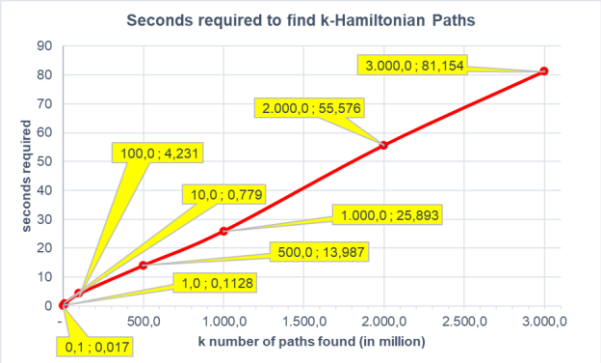

Fig. 6 Seconds required to find k-Hamiltonian paths (Developed by the authors)

The dynamic programing-based method for constructing complete Hamiltonian paths proves to be successful in efficiently finding a substantial number of paths, in contrast to searching them all simultaneously on a large TREB graph.

The following images provide visualizations of some formed Hamiltonian paths, where the node labels are filled with numbers indicating the sequence of pathway discovery from node to node. The first two images depict pathways of Hamiltonian paths formed from H00, where in Figure 7, it starts from H00 to H01, H11, and H10, and vice versa in Figure 8, the pathway goes from H00 to H10, H11, and H01.

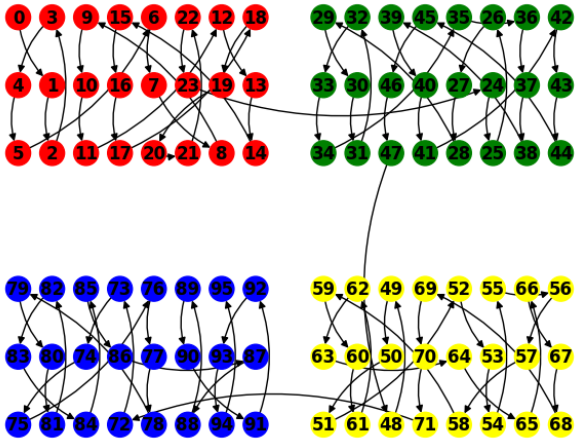

Fig. 7 A Hamiltonian path starts at H00 (Developed by the authors)

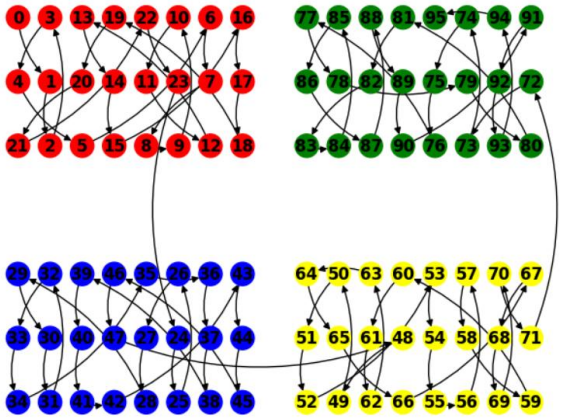

Fig. 8 Another Hamiltonian path with a different direction starts at H00 (Developed by the authors)

Similarly, two examples of Hamiltonian path search results from the initial H01 to H00, H10, and H11 and from H01 to H11, H10, and H00 are given in Figures 9 and 10, respectively.

Next, two Hamiltonian paths formed from H10 to H00, H01, and H11 and from H10 to H11, H01, and H00 are given in Figures 11 and 12, respectively.

Two examples of Hamiltonian paths found from the initial subgraph H11 to H10, H00, and H01 and from the initial subgraph H11 to H01, H00, and H10 are given in Figures 13 and 14, respectively.

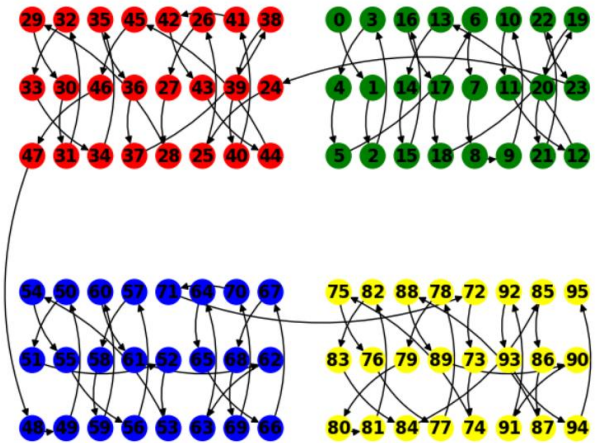

Fig. 9 A Hamiltonian path starts from H01 paths (Developed by the authors)

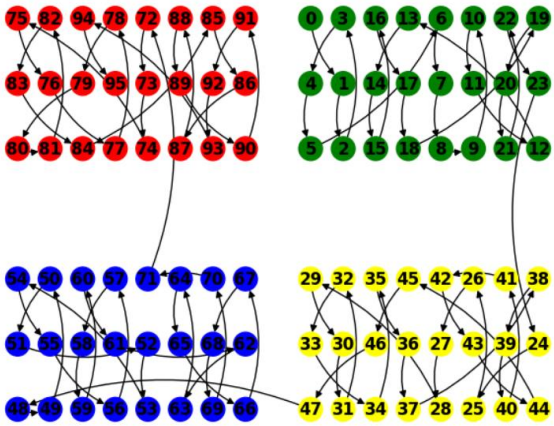

Fig. 10 Another Hamiltonian path with a different direction starts from H01 paths (Developed by the authors)

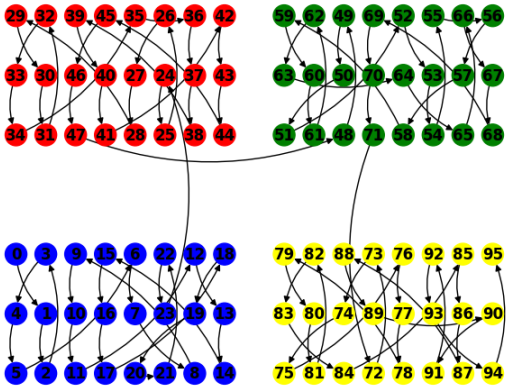

Fig. 11 A Hamiltonian path starts from Sub 10 paths (Developed by the authors)

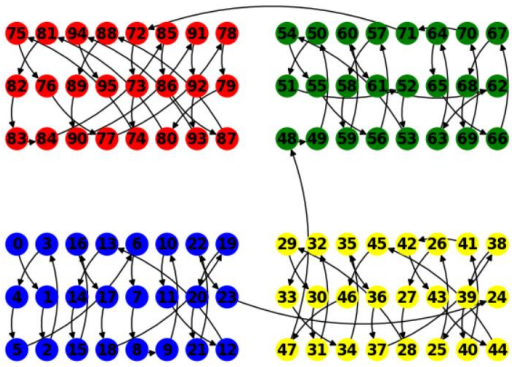

Fig. 12 A Hamiltonian path starts from H10 paths (Developed by the authors)

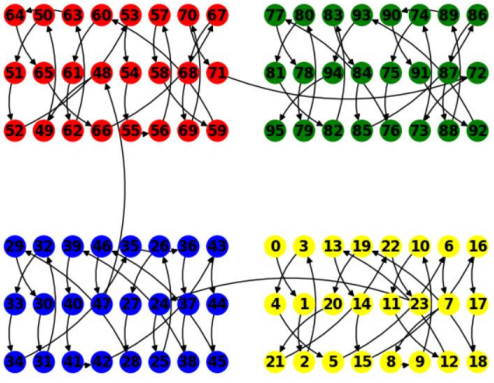

Fig. 13 A Hamiltonian path starts from H11 (Developed by the authors)

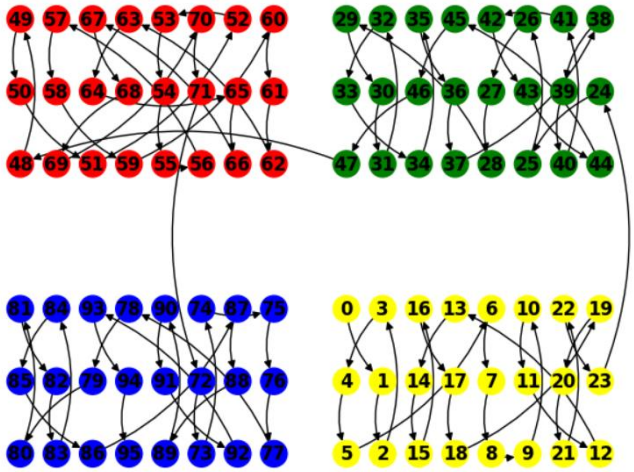

Fig. 14 Another Hamiltonian path with a different direction starts from H11 (Developed by the authors)

After discovering 36,470 Hamiltonian paths in each Hxx, to observe the performance of the *Find_kPaths* procedure, the paths in H00 are divided into 365 partitions, with 100 paths per partition (except the last partition that contains 70 paths). Similar to the previous experiment, the process involves connecting each path in H00 to paths in H01, then to paths in H11, and finally to paths within H10.

As a result, the average time required to form *k* paths in each partition is about 40 seconds. Therefore, for a total of 365 partitions, it takes about 14,600 seconds, or about 4.05 hours to finish. From an initial path in H00, a total of 36,470 Hamiltonian paths were successfully formed in the TREB graph, thus resulting in a grand total of 1,330,060,900 Hamiltonian paths in the large TREB graph from all 36,470 starting paths in H00. However, these results are from one permutation of Hxx, such as H00, H01, H11, H10, and there are additional permutations where Hxx is adjacent, such as H00, H10, H11, H01, or its reverse H01, H11, H10, H00, and so forth. In other words, the total number of Hamiltonian paths in the TREB graph is the product of the number of valid permutations of Hxx and the total number of paths already found.

5. Conclusion

This study demonstrates that, while it has been mathematically proven that there are Hamiltonian paths in the large TREB graph, from an algorithmic perspective, it not only confirms the existence of Hamiltonian paths but also reveals the pathways of their formations and the total number of Hamiltonian paths that can be generated by the proposed hybrid-based program, which combines brute force and dynamic programing algorithms. Both algorithms are part of the exact approach and are used in this study to ensure that the discovered paths are accurate and represent optimal solutions.

The exact method for path searching can still be further developed in future works to be more time-efficient. For example, by initiating path formation from the bridge to paths in both connected subgraphs.

References

[1] ADEBAYO AO, CHAUBEY M, & NUMBU LP. Industry 4.0: The Fourth Industrial Revolution and How It Relates to the Application of Internet of Things(IoT). *Journal of Multidisciplinary Engineering Science Studies*, 2019, 5(2): 2477-2482. http://dx.doi.org/10.13140/RG.2.2.13334.40008

[2] SASIAIN J, SANZ A, ASTORGA J, & JACOB E. Towards Flexible Integration of 5G and IIoT Technologies in Industry 4.0: A Practical Use Case. *Applied Sciences*, 2020; 10(21): 7670. https://doi.org/10.3390/app10217670

[3] AL FAISAL F, RAHMAN M M H, and INOGUCHI Y. An Extensive Power and Performance Analysis for High Dimensional Mesh and Torus Interconnection Networks. *International Journal of Distributed Systems and Technologies* 2023, 14(1): 1-19, https://doi.org/10.4018/IJDST.321208.

[4] AHMAD K, and SETHI M. Review of Network on Chip Routing Algorithms. *EAI Endorsed Transactions on Context-Aware Systems and Applications*, 2020, 7(22): e5, https://doi.org/10.4108/eai.23-12-2020.167793.

[5] REDDYV P, JENA S, & PRASAD V K. An Efficient Dynamic Parallel and Distributed Network with Hybrid Hyper Cube. *International Journal of Advanced Trends in Computer Science and Engineering*, 2020, 9(4): 5200-5205, https://doi.org/10.30534/ijatcse/2020/147942020.

[6] KALEEM M, and BIN ISNIN I F. A Survey on Network on Chip Routing Algorithms Criteria. In *Advances in Intelligent Systems and Computing*, 2021, 1188: 455-466 https://doi.org/10.1007/978-981-15-6048-4_40.

[7] DORAISAMY R, MOHARIR M, and ARUL R. Congestion aware and game based odd even adaptive routing in network on chip many-core architecture. *Indonesian Journal of Electrical Engineering and Computer Science*, 2022, 28(2): 962-972, https://doi.org/10.11591/ijeecs.v28.i2.pp962-972.

[8] ABRAHAM J, and AROCKIARAJ M. Minimum Linear Arrangement of the Cartesian Product of Optimal Order Graph and Path. *Parallel Processing Leters.*, 2021, 31(1): 2150004, https://doi.org/10.1142/S0129626421500043.

[9] AROCKIARAJ M, LIU J B, DELAILA J N, and SHALINI A J. On the optimal layout of balanced complete multipartite graphs into grids and tree related structures. *Discrete Applied Mathematics*, 2021, 288: 50-65, https://doi.org/10.1016/j.dam.2020.08.022.

[10] LI F, WANG W, XU Z, and ZHAO H. Some results on the lexicographic product of vertex-transitive graphs. *Applied Mathematics Letters*, 2011, 24(11): 1924-1926, https://doi.org/10.1016/j.aml.2011.05.021.

[11] ZHANG Z, XIAO W J, and WEI W H. Some properties of cartesian product of cayley graphs. *Proceedings of the 2009 International Conference on Machine Learning and Cybernetics*, 2009: 2153-2157. https://doi.org/10.1109/ICMLC.2009.5212231.

[12] PISANSKI T, and TUCKER T W. Growth in products of graphs. *Australasian Journal of Combinatorics, 2002, 26: 155-169*

[13] AMBULGEKAR S, MALEWADIKAR S, GARANDE R, and JOSHI B. Next Words Prediction Using Recurrent NeuralNetworks. *ITM Web Conferences*, 2021, 40: 03034, https://doi.org/10.1051/itmconf/20214003034.

[14] LAI P L, and TSAI C H. Embedding of tori and grids into twisted cubes. *Theoretical Computer Science*, 2010, 411(40-42): 3763-3773, https://doi.org/10.1016/j.tcs.2010.06.029.

[15] XUE S, DENG Q, LI P, and CHEN J. Hamiltonian paths and Hamiltonian cycles passing through prescribed linear forests in star graph with fault-tolerant edges. *Discrete Applied Mathematics*, 2023, 334: 68-80, https://doi.org/10.1016/j.dam.2023.02.016.

[16] PARK J H. Torus-like graphs and their paired many-to-many disjoint path covers. *Discrete Applied Mathematics*, 2021, 289: 64-77, https://doi.org/10.1016/j.dam.2020.09.008.

[17] SHVALB N, FRENKEL M, SHOVAL S, and BORMASHENKO E. Universe as a Graph (Ramsey Approach to Analysis of Physical Systems). *World Journal of Physics*, 2023, 01(01): 1-24, https://doi.org/10.56439/wjp/2023.1101.

[18] LANEL G H J, PALLAGE H K, RATNAYAKE J K, et al. A survey on Hamiltonicity in Cayley graphs and digraphs on different groups. *Discrete Mathematics, Algorithms and Applications*, 2019, 11(5): 1930002, https://doi.org/10.1142/S1793830919300029.

[19] GU H, XIE Q, WANG K, et al. X-torus: A variation of torus topology with lower diameter and larger bisection width. in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006: 149-157. https://doi.org/10.1007/11751649_16.

[20] RAHMAN M M H, INOGUCHI Y, AL FAISAL F, and KUNDU M K. Symmetric and folded tori connected torus network. *Journal of Networks*, 2011, 6(1): 26-35, https://doi.org/10.4304/jnw.6.1.26-35.

[21] LIU Y, HAN J, and DU H. A hypercube-based scalable interconnection network for massively parallel computing. *Journal of Computers*, 2008, 3: 58-65. https://doi.org/10.4304/jcp.3.10.58-65.

[22] CAI Z, XIAO W, ZHANG Q, and LIU Y. Principle of Symmetry for Network Topology with Applications to Some Networks. *Journal of Networks*, 2010, 5(9): 994-1000, https://doi.org/10.4304/jnw.5.9.994-1000.

[23] CURRAN S J, and GALLIAN J A. Hamiltonian cycles and paths in Cayley graphs and digraphs - A survey. *Discrete Mathematics*, 1996, 156(1-3): 1-18, https://doi.org/10.1016/0012-365X(95)00072-5.

[24] WITTE D, and GALLIAN J A. A survey: Hamiltonian cycles in Cayley graphs. *Discrete Mathematics*, 1984, 51(3): 293-304, https://doi.org/10.1016/0012-365X(84)90010-4.

[25] CHENG S, ZHONG W, ISAACS K E, and MUELLER K. Visualizing the Topology and Data Traffic of Multi-Dimensional Torus Interconnect Networks. *IEEE Access*, 2018, 6: 57191-57204, https://doi.org/10.1109/ACCESS.2018.2872344.

[26] CHOPKAR P N, and GAIKWAD M A. Review of XY routing algorithm for 2D torus topology of NoC architecture. *International Journal of Computer Applications Special Issue Recent Trends in Engineering Technologies*, 2013, 1: 22-26.

[27] KINI N G, KUMAR M S, and MRUTHYUNJAYA H S. Torus Embedded Hypercube Interconnection Network: A

Comparative Study. *International Journal of Computer Applications*, 2010, 1(4): 32-35, https://doi.org/10.5120/106-217.
[28] BHARDWAJ M. C2 Torus New Interconnection Network Topology Based on 2D Torus. *American Journal of Networks and Communications*, 2015, 4(3): 1-4, https://doi.org/10.11648/j.ajnc.s.2015040301.11.
[29] KINI N G, KUMAR M S, and MRUTHYUNJAYA H S. A torus embedded hypercube scalable interconnection network for parallel architecture. *Proceedings of the 2009 IEEE International Advance Computing Conference*, 2009: 858-861. https://doi.org/10.1109/IADCC.2009.4809127.
[30] GUZIDE O, and WAGH M D. Enhanced Butterfly: A Cayley Graph with Node Degree 5. *Proceedings of the 20th International Conference on Parallel and Distributed Computing Systems*, 2007: 224-229.
[31] LATIFAH L, ERNASTUTI E, and KERAMI D. Embeddings on Torus-Butterfly Interconnection Network. *International Journal of Applied Information Systems*, 2012, 4(9): 39-41, https://doi.org/10.5120/ijais12-450817.
[32] LATIFAH L, ERNASTUTI E, and KERAMI D. Structural Properties of Torus-Butterfly Interconnection Network. *International Journal of Computer Applications*, 2012, 46(16): 31-35.
[33] UMA S, and MAHESWARI B. Some Properties of Cartesian Product Graphs of Cayley Graphs with Arithmetic Graphs. *International Journal of Computer Applications*, 2016, 138(3): 26-29, https://doi.org/10.5120/ijca2016908742.
[34] ZHANG Z, and XIAO W. A new family of Cayley graph interconnection networks based on wreath product and its topological properties. *Cluster Computing*, 2011, 14(4): 483-490, https://doi.org/10.1007/s10586-011-0189-0.
[35] ZHANG Z. Some Properties in Hexagonal Torus as Cayley Graph. *Communications in Computer and Information Science*, 2011, 135: 422-428. https://doi.org/10.1007/978-3-642-18134-4_68.
[36] PANDEY A, and SINGH C. Application of Graph Theory in Real Life to Develop Routes. *International Journal for Multidisciplinary Research*, 2023, 5(2): 1-7, https://doi.org/10.36948/ijfmr.2023.v05i02.1886.
[37] SINGH K, BEDI S K, and GAUR P. Identification of the most efficient algorithm to find hamiltonian path in practical conditions, *Proceedings of the 10th International Conference on Cloud Computing, Data Science and Engineering*, 2020: 38-44. https://doi.org/10.1109/Confluence47617.2020.9058283.
[38] LERA-ROMERO G, MIRANDA BRONT J J, and SOULIGNAC F J. Dynamic Programming for the Time-Dependent Traveling Salesman Problem with Time Windows. *INFORMS Journal on Computing*, 2022, 34(6): 3292-3308, https://doi.org/10.1287/ijoc.2022.1236.
[39] LU Y, BENLIC U, and WU Q. A hybrid dynamic programming and memetic algorithm to the Traveling Salesman Problem with Hotel Selection. *Computers & Operations Research*, 2018, 90: 193-207, https://doi.org/10.1016/j.cor.2017.09.008.
[40] BAIDOO E, and OPPONG S O. Solving the TSP using Traditional Computing Approach. *International Journal of Computer Applications*, 2016, 152(8): 13-19, https://doi.org/10.5120/ijca2016911906.
[41] PETTERSSON V. H. Enumerating hamiltonian cycles. *Electronic Journal Of Combinatorics*, 2014, 21(4): 4.7, https://doi.org/10.37236/4510.
[42] MASKOOKI A, and KALLIO M. A bi-criteria moving-target travelling salesman problem under uncertainty. *European Journal of Operational Research*, 2023, 309(1): 271-295, https://doi.org/10.1016/j.ejor.2023.01.009.
[43] RIYAD W A, YEE S C P, THINAKARAN R A P, and SALAM Z A B A. Comparative evaluation of numerous optimization algorithms for compiling travel salesman problem. *Journal of Advanced Research in Dynamical and Control Systems*, 2020, 12(7 Special Issue): 877-883, https://doi.org/10.5373/JARDCS/V12SP7/20202178.
[44] WAN Y, FENG C, WU K, and WANG J. Progressive Construction of k-identifiable Networks. *Proceedings of the 30th International Symposium on Quality of Service*, 2022: 1-10. https://doi.org/10.1109/IWQoS54832.2022.9812924.
[45] CLAUSEN J. Branch and bound algorithms-principles and examples. Technical Report. Department of Computer Science, University of Copenhagen, 1999, 1-30, https://doi.org/10.1.1.5.7475.
[46] CAHYANI C A M, and WIRADINATA T. Traveling Salesman Problem Multi-destination Route Recommendation System Using Genetic Algorithm and Google Maps API. *International Journal of Computer Applications*, 2023, 185(18): 35-43, https://doi.org/10.5120/ijca2023922907.
[47] PIRIM H, BAYRAKTAR E, and EKSIOGLU B. Tabu Search: A Comparative Study. in *Tabu Search*, 2008: 1-30. https://doi.org/10.5772/5637.
[48] BAJEH A O. Optimization: A Comparative Study of Genetic and Tabu Search Algorithms. *International Journal of Computer* Applications, 2011, 31(5),
[49] LIDBE A D, HAINEN A M, and JONES S L. Comparative study of simulated annealing, tabu search, and the genetic algorithm for calibration of the microsimulation model. *Simulation*, 2017, 93(1): 21-33, https://doi.org/10.1177/0037549716683028.
[50] ADEWOLE, A. P. et al. A Comparative Study of Simulated Annealing and Genetic Algorithm for Solving the Travelling Salesman Problem. I*nternational Journal of Applied Information Systems*, 2012, 4: 6-12.
[51] DUARTE A, LAGUNA M, and MARTÍ R. Tabu Search. in *Metaheuristics for Business Analytics. EURO Advanced Tutorials on Operational Research*, 2018: 85-103. https://doi.org/10.1007/978-3-319-68119-1_4.
[52] AWAD F H, AL-KUBAISI A, and MAHMOOD M. Large-scale timetabling problems with adaptive tabu search. *Journal of Intelligent Systems*, 2022, 31(1): 168-176, https://doi.org/10.1515/jisys-2022-0003.
[53] GUPTA D. Solving TSP Using Various Meta-Heuristic Algorithms, *International Journal of Recent Contributions from Engineering, Science & IT*, 2013, 1(2): 22-26. https://doi.org/10.3991/ijes.v1i2.3233.
[54] O'REGAN G. Introduction to Algorithms. In *Mathematical Foundations in Software Engineering*. 2023: 69-84. https://doi.org/10.1007/978-3-031-26212-8_4.

参考文:

[1] ADEBAYO AO、CHAUBEY M 和 NUMBU LP。工业 4.0：第四次工业革命及其与物联网(物联网)应用的关系。多学科工程科学研究报，2019，5(2)：2477-2482.

http://dx.doi.org/10.13140/RG.2.2.13334.40008
[2] SASIAIN J、SANZ A、ASTORGA J、和 JACOB E。工业 4.0 中 5G 和工业物联网技术的灵活集成：一个实际用例。应用科学，2020，10（21）：7670。https://doi.org/10.3390/app10217670
[3] AL FAISAL F、RAHMAN M M H 和 INOGUCHI Y。高维网格和环形互连网络的广泛功耗和性能分析.《国际分布式系统与技术杂志》2023，14(1)：1-19，https：//doi.org/10.4018/IJDST.321208。
[4] AHMAD K 和 SETHI M。芯片网络路由算法综述。环境影响评价支持上下文感知系统和应用程序交易，2020，7(22)：e5，https://doi.org/10.4108/eai.23-12-2020.167793。
[5] REDDYV P、JENA S 和 PRASAD V K。具有混合超立方体的高效动态并行分布式网络。国际计算机科学与工程高级趋势杂志，2020，9(4)：5200-5205，https://doi.org/10.30534/ijatcse/2020/147942020。
[6] KALEEM M 和 BIN ISNIN I F。芯片路由算法标准网络调查。智能系统和计算的进展，2021，1188：455-466 https://doi.org/10.1007/978-981-15-6048-4_40。
[7] DORAISAMY R、MOHARIR M 和 ARUL R。片上网络多核架构中的拥塞感知和基于游戏的奇偶自适应路由。印度尼西亚电气工程与计算机科学杂志，2022，28(2)：962-972，https://doi.org/10.11591/ijeecs.v28.i2.pp962-972。
[8] ABRAHAM J 和 AROCKIARAJ M。最优阶图和路径的笛卡尔积的最小线性排列。并行处理快报，2021，31(1)：2150004，https://doi.org/10.1142/S0129626421500043。
[9] AROCKIARAJ M，LIU J B，DELAILA J N，和 SHALINI A J. 关于平衡完整多部分图到网格和树相关结构的优化布局。离散应用数学，2021，288：50-65，https://doi.org/10.1016/j.dam.2020.08.022。
[10] LI F, WANG W, XU Z, 和 ZHAO H. 关于顶点传递图的字典序积的一些结果. 应用数学快报，2011，24(11)：1924-1926，https://doi.org/10.1016/j.aml.2011.05.021。
[11] ZHANG Z, XIAO W J, 和 WEI W H. 凯莱图笛卡尔积的一些性质. 2009 年机器学习和控制论国际会议论文集，2009：2153-2157。https://doi.org/10.1109/ICMLC.2009.5212231。
[12] PISANSKI T 和 TUCKER T W。图乘积的增长。澳大利亚组合学杂志，2002，26：155-169
[13] AMBULGEKAR S、MALEWADIKAR S、GARANDE R 和 JOSHI B。使用循环神经网络预测下一个单词。信息技术管理网络会议，2021，40：03034，https://doi.org/10.1051/itmconf/20214003034。
[14] LAI P L, 和 TSAI C H. 将环面和网格嵌入扭曲立方体。理论计算机科学，2010，411（40-42）：3763-3773，https://doi.org/10.1016/j.tcs.2010.06.029。
[15] XUE S，DENG Q，LI P，和 CHEN J. 通过具有容错边的星图中规定的线性森林的哈密顿路径和哈密顿循环。离散应用数学，2023，334：68-80，https://doi.org/10.1016/j.dam.2023.02.016。
[16] PARK J H. 类环图及其配对的多对多不相交路径覆盖。离散应用数学，2021，289：64-77，https://doi.org/10.1016/j.dam.2020.09.008。
[17] SHVALB N、FRENKEL M、SHOVAL S 和 BORMASHENKO E。宇宙作为图（拉姆齐物理系统分析方法）。世界物理学杂志，2023，01（01）：1-24，https://doi.org/10.56439/wjp/2023.1101。
[18] LANEL G H J，PALLAGE HK，RATNAYAKE J K，等。不同群的凯莱图和有向图中哈密顿度的调查。离散数学、算法与应用，2019，11(5)：1930002，https://doi.org/10.1142/S1793830919300029。
[19] GU H, XIE Q, WANG K, 等。X 环面：环面拓扑的一种变体，具有较小的直径和较大的二等分宽度。计算机科学讲义（包括人工智能讲义和生物信息学讲义子系列），2006：149-157。https://doi.org/10.1007/11751649_16.
[20] RAHMAN M M H、INOGUCHI Y、AL FAISAL F 和 KUNDU M K。对称和折叠环面连接环面网络。网络杂志，2011，6(1)：26-35，https://doi.org/10.4304/jnw.6.1.26-35。
[21] LIU Y, HAN J, 和 DU H. 用于大规模并行计算的基于超立方体的可扩展互连网络。计算机学报，2008，3：58-65。https://doi.org/10.4304/jcp.3.10.58-65。
[22] CAI Z, XIAO W, ZHANG Q, 和 LIU Y. 网络拓扑对称原理及其在某些网络中的应用。网络杂志，2010，5(9)：994-1000，https://doi.org/10.4304/jnw.5.9.994-1000。
[23] CURRAN S J 和 GALLIAN J A. 凯莱图和有向图中的哈密顿循环和路径-一项调查。离散数学，1996，156(1-3)：1-18，https://doi.org/10.1016/0012-365X(95)00072-5.
[24] WITTE D 和 GALLIAN J A。一项调查：凯莱图中的哈密顿循环。离散数学，1984，51(3)：293-304，

https://doi.org/10.1016/0012-365X(84)90010-4。

[25] CHENG S、ZHONG W、ISAACS K E 和 MUELLER K。可视化多维环面互连网络的拓扑和数据流量。IEEE 访问，2018，6：57191-57204，https://doi.org/10.1109/ACCESS.2018.2872344。

[26] CHOPKAR P N 和 GAIKWAD M A。片上网络架构 2D 环面拓扑的 XY 路由算法综述。国际计算机应用杂志特刊工程技术最新趋势，2013，1：22-26。

[27] KINI NG、KUMAR MS 和 MRUTHYUNJAYA H S。环面嵌入式超立方体互连网络：比较研究。国际计算机应用杂志，2010，1(4)：32-35，https://doi.org/10.5120/106-217。

[28] BHARDWAJ M. C2 环面基于 2D 环面的新型互连网络拓扑。美国网络与通信杂志，2015，4(3)：1-4，https://doi.org/10.11648/j.ajnc.s.2015040301.11。

[29] KINI NG、KUMAR MS 和 MRUTHYUNJAYA H S。用于并行架构的环形嵌入式超立方体可扩展互连网络。2009 年 IEEE 国际高级计算会议论文集，2009：858-861。https://doi.org/10.1109/IADCC.2009.4809127。

[30] GUZIDE O 和 WAGH MD。增强型蝴蝶：节点度为 5 的凯莱图。第 20 届并行和分布式计算系统国际会议论文集，2007：224-229。

[31] LATIFAH L、ERNASTUTI E 和 KERAMI D。环形蝴蝶互连网络上的嵌入。国际应用信息系统杂志，2012，4(9)：39-41，https://doi.org/10.5120/ijais12-450817。

[32] LATIFAH L、ERNASTUTI E 和 KERAMI D。环形蝴蝶互连网络的结构特性。国际计算机应用杂志，2012，46（16）：31-35。

[33] UMA S 和 MAHESWRI B。带有算术图的凯莱图的笛卡尔积图的一些性质。国际计算机应用杂志，2016，138(3)：26-29，https://doi.org/10.5120/ijca2016908742。

[34] ZHANG Z, 和 XIAO W. 基于环积及其拓扑性质的凯莱图互连网络新族. 集群计算，2011，14(4)：483-490，https://doi.org/10.1007/s10586-011-0189-0。

[35] ZHANG Z. 六角环面作为凯莱图的一些性质。计算机与信息科学通信，2011，135：422-428。https://doi.org/10.1007/978-3-642-18134-4_68。

[36] PANDEY A 和 SINGH C。图论在现实生活中应用以开发路线。国际多学科研究杂志，2023，5(2)：1-7，https://doi.org/10.36948/ijfmr.2023.v05i02.1886。

[37] SINGH K，BEDI S K，和 GAUR P。在实际条件下寻找哈密尔顿路径的最有效算法的识别，第十届云计算、数据科学与工程国际会议论文集，2020：38-44。https://doi.org/10.1109/Confluence47617.2020.9058283。

[38] LERA-ROMERO G、MIRANDA BRONT J J 和 SOULIGNAC F J。具有时间窗的依赖时间的旅行商问题的动态规划。通知计算杂志，2022，34(6)：3292-3308，https://doi.org/10.1287/ijoc.2022.1236。

[39] LU Y, BENLIC U, 和 WU Q. 针对酒店选择的旅行商问题的混合动态规划和模因算法。计算机与运筹学，2018，90：193-207，https://doi.org/10.1016/j.cor.2017.09.008。

[40] BAIDOO E 和 OPPOONG S O。使用传统计算方法求解总磷。国际计算机应用杂志，2016，152(8)：13-19，https://doi.org/10.5120/ijca2016911906。

[41] PETTERSSON V. H. 枚举哈密顿循环。组合学电子杂志，2014，21(4)：4.7，https://doi.org/10.37236/4510。

[42] MASKOKKI A 和 KALLIO M。不确定性下的双标准移动目标旅行推销员问题。欧洲运筹学杂志，2023，309(1)：271-295，https://doi.org/10.1016/j.ejor.2023.01.009。

[43] RIYAD W A、YEE S C P、THINAKARAN R A P 和 SALAM Z A B A。编译旅行推销员问题的多种优化算法的比较评估。动力与控制系统高级研究杂志，2020，12（7 号特刊）：877-883，https://doi.org/10.5373/JARDCS/V12SP7/20202178。

[44] WAN Y, FENG C, WU K, 和 WANG J. k-可识别网络的渐进式构建。2022 年第 30 届服务质量国际研讨会论文集 2022：1-10。https://doi.org/10.1109/IWQoS54832.2022.9812924。

[45] CLAUSEN J. 分支定界算法-原理和示例。技术报告。哥本哈根大学计算机科学系，1999，1-30，https://doi.org/10.1.1.5.7475。

[46] CAHYANI C A M 和 WIRADINATA T. 使用遗传算法和谷歌地图应用程序编程接口的旅行商问题多目的地路线推荐系统。国际计算机应用杂志，2023，185(18)：35-43，https://doi.org/10.5120/ijca2023922907。

[47] PIRIM H、BAYRAKTAR E 和 EKSIOGLU B. 禁忌搜索：比较研究。载于《禁忌搜索》，2008：1-30。https://doi.org/10.5772/5637。

[48] BAJEH A O。优化：遗传和禁忌搜索算法的比较研究。国际计算机应用杂志，2011，31（5），

[49] LIDBE A D、HAINEN A M 和 JONES S L。模拟退火、禁忌搜索和用于校准微观模拟模型的遗传算法的比较研究。模拟，2017，93(1)：21-33，https://doi.org/10.1177/0037549716683028。

[50] ADEWOLE, A. P. 等。模拟退火与遗传算法求解旅行商问题的比较研究。国际应用信息系统杂志，2012，4：6-12。

[51] DUARTE A、LAGUNA M 和 MARTÍ R. 禁忌搜索。商业分析元启发法。欧洲运筹学高级教程，2018：85-103。https://doi.org/10.1007/978-3-319-68119-1_4。

[52] AWAD F H、AL-KUBAISI A 和 MAHMOOD M。自适应禁忌搜索的大规模时间表问题。智能系统杂志，2022，31(1)：168-176，https://doi.org/10.1515/jisys-2022-0003。

[53] GUPTA D. 使用各种元启发式算法求解总磷，国际工程、科学与它近期贡献期刊，2013，1(2)：22-26。https://doi.org/10.3991/ijes.v1i2.3233。

[54] O'REGAN G. 算法导论。软件工程的数学基础。2023：69-84。https://doi.org/10.1007/978-3-031-26212-8_4。