sensors

Article

Torus Pairwise Disjoint-Path Routing †

Antoine Bossard [1,*] and Keiichi Kaneko [2]

1 Graduate School of Science, Kanagawa University, Kanagawa 259-1293, Japan
2 Graduate School of Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan; k1kaneko@cc.tuat.ac.jp
* Correspondence: abossard@kanagawa-u.ac.jp

† This paper is an extended version of our paper: Bossard, A.; Kaneko, K. On the torus pairwise disjoint-path routing problem. In Proceedings of the 18th IEEE International Conference on Computer and Information Technology (CIT), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1739–1746.

Received: 27 September 2018; Accepted: 3 November 2018; Published: 13 November 2018

Abstract: Modern supercomputers include hundreds of thousands of processors and they are thus massively parallel systems. The interconnection network of a system is in charge of mutually connecting these processors. Recently, the torus has become a very popular interconnection network topology. For example, the Fujitsu K, IBM Blue Gene/L, IBM Blue Gene/P, and Cray Titan supercomputers all rely on this topology. The pairwise disjoint-path routing problem in a torus network is addressed in this paper. This fundamental problem consists of the selection of mutually vertex disjoint paths between given vertex pairs. Proposing a solution to this problem has critical implications, such as increased system dependability and more efficient data transfers, and provides concrete implementation of green and sustainable computing as well as security, privacy, and trust, for instance, for the Internet of Things (IoT). Then, the correctness and complexities of the proposed routing algorithm are formally established. Precisely, in an n-dimensional k-ary torus ($n < k$, $k \geq 5$), the proposed algorithm connects c ($c \leq n$) vertex pairs with mutually vertex-disjoint paths of lengths at most $2k(c-1) + n\lfloor k/2 \rfloor$, and the worst-case time complexity of the algorithm is $O(nc^4)$. Finally, empirical evaluation of the proposed algorithm is conducted in order to inspect its practical behavior.

Keywords: parallel processing; interconnect; system dependability; fault tolerance; algorithm

1. Introduction

Since the development of parallel supercomputers, the number of included processors has been continuously rising. The hardware component that is responsible for the connection of processors is the interconnection network (a.k.a., interconnect). Hypercubes [1] were a popular topology for the interconnection network of massively parallel systems in the eighties. Intel developed, for instance, the iPSC supercomputer series, with the iPSC/1 device connecting 32 to 128 cores: the iPSC/d5 is based on a five-dimensional hypercube, hence connecting $2^5 = 32$ cores; the iPSC/6 is based on a six-dimensional hypercube, hence connecting 64 cores, and the iPSC/d7 128 cores. A similar approach was followed by the nCUBE company that built, for example, the nCUBE 10 device, which is based on a 10-dimensional hypercube, hence connecting 1024 cores.

Nowadays, machines of the megaFLOPs era, such as the nCUBE, have been replaced by ones featuring a computing power of several petaFLOPs, and with even the exaFLOP barrier possibly being reached as soon as 2020 by the Cray company [2]. Regarding the number of processors embodied, modern massively parallel systems rely on hundreds of thousands of them. One step ahead, the Sunway TaihuLight supercomputer, ranked world number one in June 2017 and second as of June 2018 [3], includes more than one million cores. Considering this very large number of processors, it is easy to understand that data communication efficiency and interconnection networks in general are critical to achieving the highest computing performance. Indeed, in the case of suboptimal core interconnection and data transfers, bottleneck situations would inevitably arise, inducing underused cores.

Just as complete networks are not practical as interconnection networks of massively parallel systems given the prohibitively high number of edges per processor induced, hypercubes are no more a solution considering the number of nodes involved. Precisely, the vertex degree in the case of a hypercube becomes rapidly impractical as n edges per vertex are required for the hypercube to connect, in total, 2^n vertices. As a result, various topologies designed for interconnection networks have been introduced. For example, Li et al. proposed the dual-cube [4] and metacube [5] topologies, both based on hypercubes. The star-graph [6] is another topology example, this time based on permutations, which was used to introduce other topologies, such as burnt pancake graphs [7].

Topologies based on meshes [8] are another solution for interconnection networks. One of these, the torus topology [9], is addressed in this paper. Thanks to a simple definition and an advantageous network order compared with, for instance, hypercubes, the torus network topology is very popular as interconnect of supercomputers. Indeed, numerous machines listed in the TOP500 world ranking are based on the torus topology. For example, the Fujitsu K (Tofu interconnect [10,11]), IBM Blue Gene/L, IBM Blue Gene/P, and Cray Titan (Gemini interconnect [12]) supercomputers [3]. The main topological properties of tori and those of various other interconnection networks are given in Table 1. In this table, the network cost is the product of network degree by network diameter.

Table 1. Comparing torus topological properties with other popular interconnection networks [9].

Network	Order	Degree	Diameter	Cost	Links
(n, k)-torus	k^n	$2n$	$n\lfloor k/2\rfloor$	$2n^2\lfloor k/2\rfloor$	nk^n
n-hypercube	2^n	n	n	n^2	$2^n n$
n-dual-cube	2^{2n-1}	n	$2n$	$2n^2$	$2^{2n-2}n$
n-star graph	$n!$	$n-1$	$\lfloor 3(n-1)/2\rfloor$	$(n-1)\lfloor 3(n-1)/2\rfloor$	$n!(n-1)/2$
(k, m)-metacube	$2^{2^k m+k}$	$m+k$	$2^k(m+1)$	$2^k(m+1)(m+k)$	$2^{2^k m+k-1}(m+k)$

In this paper, the torus pairwise disjoint-path routing problem is addressed. This critical data communication problem consists of the selection of mutually vertex-disjoint paths between several vertex pairs. As detailed next, this problem has numerous applications. Disjoint-path routing in general is a very desirable property for routing algorithms, and this is for several reasons. First, disjoint-path routing enables simultaneous data transfers over the network, and for that, without the need to change switching patterns inside routers, on the one hand optimizing data transfer performance, and, on the other hand, reducing network usage time. While the former consequence has obvious positive implications, the latter thus induces concrete implementation of energy-harvesting communications and networks, which contents green and sustainable computing [13,14], with applications, for instance, to the Internet of Things (IoT) and cyber–physical systems. In addition, path disjointness enforced at the hardware level with circuit switching means that an optimum degree of privacy is achieved as a data transfer is never interrupted by another transmission. This research thus directly addresses the issues of security, privacy, and trust for the IoT. Second, disjoint-path routing importantly ensures that the notorious blocking situations of parallel processing, deadlocks, livelocks and starvations, never occur, thus facilitating, for instance, distributed data processing in sensor applications. Third, not only efficiency but, by selecting disjoint paths, system dependability is also significantly increased. Effectively, where multiple nondisjoint paths could be rendered useless by a single faulty vertex (i.e., the faulty vertex is common to several paths), disjoint paths are much more robust: one faulty vertex can jeopardize, at most, one path, since any vertex of the network is included in at most one of the selected paths. Such robustness has positive implications regarding, for instance, the quality of experience and service in the IoT and cyber–physical systems.

In an arbitrary c-connected graph $G(V, E)$, the existence of c disjoint paths for each of the node-to-node disjoint paths, the node-to-set disjoint paths, and the set-to-set disjoint paths problems are ensured by Menger's theorem [15]. In fact, disjoint paths can be obtained by applying the maximum-flow algorithm assigning unit capacity to each of the edges and the vertices. For c pairs of vertices in arbitrary graph $G(V, E)$, the problem whether there are c disjoint paths between the c pairs is NP complete if c is a variable of the problem input [16]. For any fixed c, the problem can be solved in $O(|V|^3)$ [17] though it is still a hard problem.

Due to higher complexity, the pairwise disjoint-path routing problem is typically addressed last after introducing algorithms that solve the unicast (a.k.a., point-to-point, one-to-one, or node-to-node), node-to-node disjoint-path, node-to-set disjoint-path and set-to-set disjoint-path routing problems. In an n-dimensional k-ary torus, a node-to-node routing algorithm has been described in Reference [18] and with fault tolerance in Reference [19], the latter selecting paths of lengths at most $n\lfloor k/2 \rfloor + 1$, with a worst-case time complexity of $O(k^2)$. A node-to-set disjoint-path routing algorithm in a torus has also been described in Reference [19], with paths of lengths at most $n\lfloor k/2 \rfloor + 1$, and a worst-case time complexity of $O(k^3)$. A torus set-to-set disjoint-path routing algorithm has been given in Reference [20], with paths of lengths at most $2(k+1)n$, and a worst-case time complexity of $O(kn^3 + n^3 \log n)$.

As for different topologies, Gu and Peng described a pairwise disjoint-path routing algorithm in a hypercube [21] and in a star-graph [22]. Bossard and Kaneko solved the same problem in perfect hierarchical hypercubes [23], as did Sawada et al. in pancake graphs [24], and Park focused on restricted hypercube-like graphs [25]. The approach in this paper to select mutually vertex-disjoint paths is, as used in several previous works, to rely on the recursive structure of a torus. Precisely, in an n-dimensional k-ary torus, given c ($c \leq n$) vertex pairs, the algorithm proposed here selects c mutually vertex-disjoint paths of lengths at most $2k(c-1) + n\lfloor k/2 \rfloor$ with a worst-case time complexity of $O(nc^4)$.

Related research on disjoint-path routing can be found in Reference [26] where it is applied to data-center networks. Close to disjoint-path routing, independent spanning tree construction has been researched, for instance, in Möbius cubes in [27]. Other related works include the calculation of various topological values of the network topology, such as topological indices [28] and the least eigenvalue [29].

The rest of this paper is organized as follows. Definitions, notations and intermediary results are established in Section 2. The proposed routing algorithm is described in Section 3 and exemplified in Section 4. The proof of the algorithm correctness, including the fact that the selected paths are disjoint, is given in Section 5, which relies on lemmas proved in Section 2. Complexities are formally established in Section 6 and the proposed algorithm is empirically verified and evaluated in Section 7. Finally, this paper is concluded in Section 8.

2. Preliminaries

For the sake of clarity, a vertex pair (in this paper, typically a source-destination vertex pair), denoted by (u, v), can be considered as the set $\{u, v\}$. For vertex u, define set $N(u)$ as the set of the neighbor vertices of u.

In a graph, a path p is an alternate sequence of vertices and links $u_1, (u_1, u_2), u_2, \ldots, (u_{n-1}, u_n), u_n$. Path p can be similarly written as $u_1 \to u_2 \to \ldots \to u_n$ and abbreviated as $u_1 \rightsquigarrow u_n$ when explicit mention of the vertices between u_1 and u_n is not required. The length of a path is defined as its number of links, hence p has length $n-1$. Two paths are mutually vertex-disjoint (simply *disjoint* hereinafter) if and only if they have no vertex in common.

Definition 1 ([8])**.** *An n-dimensional k-ary torus, denoted by (n, k)-torus, is an undirected graph made of the k^n vertices induced by the set $\{0, 1, \ldots, k-1\}^n$. Two vertices $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$ of an (n, k)-torus are adjacent if and only if* $\exists j\ (1 \leq j \leq n), \forall i\ (1 \leq i \leq n, i \neq j), a_i = b_i, a_j = (b_j \pm 1) \bmod k$.

Thus, a vertex of an $(n,\ k)$-torus is an n-dimensional vector. Therefore, two vertices of an (n, k)-torus can be compared (i.e., equality testing) in linear $O(n)$ time, by simply going through each of all the n coordinates of the two vertices. A (2, 4)-torus is illustrated in Figure 1.

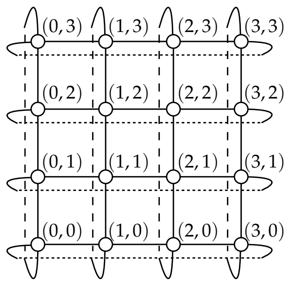

Figure 1. A two-dimensional four-ary torus, with vertex addresses in the top right-hand corner of each vertex [9].

An $(n,\ k)$-torus is a recursive topology: for one dimension δ $(1 \leq \delta \leq n)$, it consists of k $(n-1, k)$-tori (called *subtori*). For instance, considering the horizontal dimension of the (2, 4)-torus of Figure 1, that is the dimension $\delta = 1$, this torus consists of four (1, 4)-tori: these subtori appear vertically in the figure. It is said that a path goes through a subtorus T if and only if it includes a vertex of T.

Next, operator γ is defined for convenient vertex coordinate manipulation. In an $(n,\ k)$-torus, for a vertex $u = (u_1, u_2, \ldots, u_n)$ and a dimension δ $(1 \leq \delta \leq n)$, let $\gamma(u, \delta)$ denote the δ coordinate of vertex u. For instance, $\gamma(u, 4) = u_4$. This definition is extended to subtori: let $\gamma(T, \delta)$ denote the δ coordinate of subtorus T. For instance, considering the torus of Figure 1 and the dimension $\delta = 1$, we have $\gamma(T, \delta) = 0$ where T is the leftmost subtorus (i.e., T consists of the vertices $(0, 0)$, $(0, 1)$, $(0, 2)$ and $(0, 3)$). Therefore, for dimension δ, a subtorus T can be unambiguously specified by the value of $\gamma(T, \delta)$.

Then, still considering vertex u and dimension δ, candidate paths and the corresponding path sets for traversing the dimension δ are defined as follows. Note that these paths specify how to traverse a dimension, thus remaining "open" (i.e., with a start vertex, but no end vertex) for the sake of simplicity.

Definition 2. *Given two vertices* $a = (a_1, a_2, \ldots, a_n)$ *and* $b = (b_1, b_2, \ldots, b_n)$ *of an* $(n,\ k)$*-torus, define* $a \oplus b = ((a_1 + b_1) \bmod k, (a_2 + b_2) \bmod k, \ldots, (a_n + b_n) \bmod k)$ *and* $a \ominus b = ((a_1 - b_1) \bmod k, (a_2 - b_2) \bmod k, \ldots, (a_n - b_n) \bmod k)$*.*

Definition 3. *n unit vectors e_i $(1 \leq i \leq n)$ are defined as $(u_1, u_2, \ldots, u_n)$ where $u_j = 0$ $(1 \leq j \leq n, i \neq j)$ and $u_i = 1$.*

First, define the path:

$$p^+(u, \delta): \quad u \to u \oplus e_\delta \to (u \oplus e_\delta) \oplus e_\delta \to \ldots$$

which can be similarly written as $u \to (u_1, u_2, \ldots, (u_\delta + 1) \bmod k, \ldots, u_n) \to (u_1, u_2, \ldots, (u_\delta + 2) \bmod k, \ldots, u_n) \to \ldots$

Next, define the two path sets:

$$P_1^+(u, \delta): \quad \bigcup_{\substack{1 \leq i \leq n \\ i \neq \delta}} \left\{ \begin{array}{l} u \to u \oplus e_i \to (u \oplus e_i) \oplus e_\delta \to ((u \oplus e_i) \oplus e_\delta) \oplus e_\delta \to \ldots, \\ u \to u \ominus e_i \to (u \ominus e_i) \oplus e_\delta \to ((u \ominus e_i) \oplus e_\delta) \oplus e_\delta \to \ldots \end{array} \right\}$$

and

$$P_2^+(u,\delta): \quad \bigcup_{\substack{1\le i\le n\\ i\neq\delta}} \left\{ \begin{array}{l} u \to u\oplus e_i \to (u\oplus e_i)\oplus e_i \to ((u\oplus e_i)\oplus e_i)\oplus e_\delta \\ \quad \to (((u\oplus e_i)\oplus e_i)\oplus e_\delta)\oplus e_\delta \to \ldots, \\ u \to u\ominus e_i \to (u\ominus e_i)\ominus e_i \to ((u\ominus e_i)\ominus e_i)\oplus e_\delta \\ \quad \to (((u\ominus e_i)\ominus e_i)\oplus e_\delta)\oplus e_\delta \to \ldots \end{array} \right\}$$

Path $p^-(u,\delta)$ and path sets $P_1^-(u,\delta)$, $P_2^-(u,\delta)$ are defined similarly but with $\ominus\ e_\delta$ instead of $\oplus\ e_\delta$ (details in the appendix). In other words, path $p^+(u,\delta)$ (resp. $p^-(u,\delta)$) makes no detour when traversing dimension δ, while the paths of $P_1^+(u,\delta)$ and $P_2^+(u,\delta)$ (resp. $P_1^-(u,\delta)$ and $P_2^-(u,\delta)$) make a detour of one and two links, respectively.

Finally, define the two path sets:

$$P^+(u,\delta) = \{p^+(u,\delta)\} \cup P_1^+(u,\delta) \cup P_2^+(u,\delta)$$

and

$$P^-(u,\delta) = \{p^-(u,\delta)\} \cup P_1^-(u,\delta) \cup P_2^-(u,\delta)$$

The paths of $P^+(u,\delta)$ and $P^-(u,\delta)$ in the case of a (3, 5)-torus are illustrated in Figure 2, where δ is the dimension used to distinguish subtori.

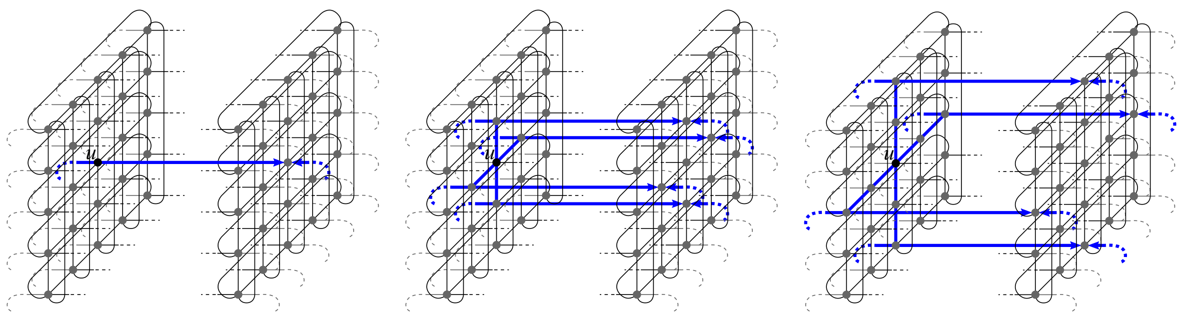

Figure 2. Considered paths (represented by arrows) for traversing the subtori of a (3, 5)-torus from the vertex u. (**a**) $p^-(u,\delta)$, $p^+(u,\delta)$, (**b**) $P_1^-(u,\delta)$, $P_1^+(u,\delta)$ and (**c**) $P_2^-(u,\delta)$, $P_2^+(u,\delta)$.

For the sake of conciseness, for torus vertex u and dimension δ, paths $p^+(u,\delta)$ and $p^-(u,\delta)$, as well as path sets $P_1^+(u,\delta)$, $P_1^-(u,\delta)$, $P_2^+(u,\delta)$ and $P_2^-(u,\delta)$ are abbreviated to $p(u,\delta)$, $P_1(u,\delta)$ and $P_2(u,\delta)$, respectively, when no ambiguity arises.

Next, a torus point-to-point routing algorithm is recalled. Unicast routing in a torus can be achieved simply with a dimension-order routing algorithm. A dimension-order routing algorithm for packet forwarding in two dimensional meshes is presented by Duato et al. [8]. It can be easily adjusted for routing in an n-dimensional torus as listed in Algorithm 1. The maximum length of a path selected by this algorithm between any two vertices is thus $n\lfloor k/2 \rfloor$.

To conclude this section, we introduce two essential lemmas on which the algorithm proposed in this paper is based (Section 3).

Lemma 1. *In an (n,k)-torus ($n \ge 2, k \ge 5$), given $c \le n$ vertex pairs (s_i, d_i) satisfying $\{s_i, d_i\} \cap \{s_j, d_j\} = \emptyset$ ($1 \le i, j \le c$, $i \neq j$) with thus $s_i = d_i$ allowed, and one subtorus T on a dimension δ ($1 \le \delta \le n$), disjoint paths (at the exception that the paths for s_i and d_i ($1 \le i \le c$) need not be disjoint) that route each of all vertices s_i, d_i to T can be found in $O(nc^2)$ time. Maximum path length is $k + 1$.*

Algorithm 1: Point-to-point simple routing in a torus with a dimension-order routing algorithm.

```
Input: Source vertex $s = (s_1, s_2, \ldots, s_n)$ and destination vertex $d = (d_1, d_2, \ldots, d_n)$ of an
    $(n, k)$-torus.
Output: A shortest path $s \rightsquigarrow d$.
$R := s$;                                   // Result initial value (path of length 0).
$u := s$;
for $i := 1$ to $n$ do
    $\Delta := d_i - s_i$;
    if $(\Delta < 0$ AND $|\Delta| < k/2)$ OR $(\Delta > 0$ AND $|\Delta| > k/2)$ then
        $\odot := \ominus$;                 // Define $\odot$ as the $\ominus$ operation.
    else
        $\odot := \oplus$;                  // Define $\odot$ as the $\oplus$ operation.
    end
    while $\gamma(u, i) \neq d_i$ do
        $u := u \odot e_i$;
        $R := R \to u$;                     // One link added to the path.
    end
end
return $R$;
```

Proof. First, assume without loss of generality that s_i is to be routed to T, and that $\gamma(s_i, \delta) \leq \gamma(T, \delta)$. The same discussion holds for d_i and for the case $\gamma(s_i, \delta) > \gamma(T, \delta)$. We show that there always remains at least one path of $P^+(s_i, \delta)$ that is not blocked by another source vertex, destination vertex or their respective paths towards T.

First, the paths of $P^+(s_i, \delta)$ consist of shortest path $p^+(s_i, \delta)$, and of the two sets $P_1^+(s_i, \delta)$ and $P_2^+(s_i, \delta)$. The paths of these two sets are not disjoint: a neighbor u of s_i is included in one single path of each of the two path sets. Hence, if $u = s_j$ $(i \neq j)$, two candidate paths are blocked by one single vertex (s_j). The reason for considering P_1 and P_2 paths is that vertex d_i can indirectly block a candidate path for s_i. Effectively, d_i may trigger the selection of a non-shortest path for a vertex s_j towards T $(i \neq j)$, and thus in total d_i and s_j each block one path for s_i. See Figure 3.

Vertex d_i on its own is not a blocker for s_i. Thus, we consider the case where d_i is an indirect blocker through vertex say s_j.

So, by considering the paths of P_1 and P_2, we ensure that either there remains one of the two paths, $(s_i \to u \to \ldots) \in P_1^+(s_i, \delta)$ and $(s_i \to u \to \ldots) \in P_2^+(s_i, \delta)$ that is not blocked by d_i, s_j, or, if not, d_i, s_j block two nondisjoint paths of $P_1^+(s_i, \delta) \cup P_2^+(s_i, \delta)$. In the first situation, we necessarily have $s_j, d_i \notin N(s_i)$, thus $p^+(s_i, \delta)$ and one path of $P_1^+(s_i, \delta)$ are blocked *but* the path $s_i \to u \to \ldots$ of $P_2^+(s_i, \delta)$ remains unblocked. In the second situation, two nondisjoint paths are blocked by d_i, s_j; thus, d_i, s_j count as one blocker for s_i (they block only one of the *disjoint* candidate paths). Therefore, it is sound to assume that d_i does not count as a blocker for s_i.

At the exception of d_i, each of all $2(n-1)$ other blockers for s_i (i.e., $(S \cup D) \setminus \{s_i, d_i\}$) can block at most one of the disjoint candidate paths. In total, it is possible to select $2(n-1)+1$ disjoint paths from s_i to T (i.e., considering the candidate paths $\{p^+(s_i, \delta)\} \cup P_1^+(s_i, \delta) \cup P_2^+(s_i, \delta)$). Therefore, considering that at most $2(n-1)$ of these disjoint paths are blocked, there always remain $[2(n-1)+1] - 2(n-1) = 1$ path to route s_i to T. See Figure 4.

Maximum path length would be obtained if s_i is routed to T with a path of P_2, hence of length $(k-1)+2 = k+1$. To route s_i to T, we first enumerate the paths of $\{p^+(s_i, \delta)\} \cup P_1^+(s_i, \delta)$ (starting with $p^+(s_i, \delta)$). If all are blocked, it means that d_i is an (indirect) blocker, thus inducing the check of at most one path of P_2 (i.e., if the first checked path of P_2 is blocked, the second one will always do).

Hence, the worst-case time complexity to route s_i to T is $O(nc)$, and, in total, $O(nc^2)$ is required to route all vertices s_i, d_i $(1 \leq i \leq c)$ to T. □

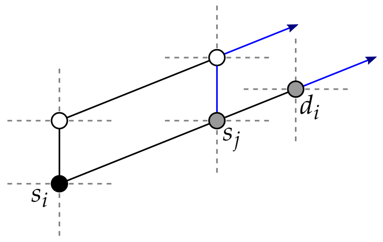

Figure 3. Vertex d_i can be indirectly a blocker for vertex s_i (selected paths in blue) [9].

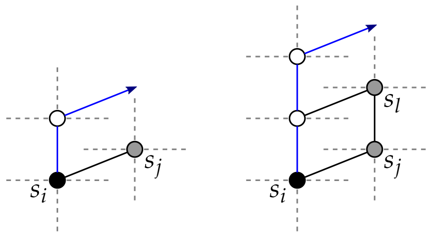

Figure 4. A vertex of $(S \cup D) \setminus \{s_i, d_i\}$ can block at most one of the disjoint candidate paths for a vertex s_i (selected paths in blue) [9].

Lemma 2. *In an (n, k)-torus ($n \geq 3$, $k \geq 5$), given $3 \leq c \leq n$ vertex pairs (s_i, d_i) satisfying $\{s_i, d_i\} \cap \{s_j, d_j\} = \emptyset$ $(1 \leq i, j \leq c,\ i \neq j)$ with thus $s_i = d_i$ allowed, and two subtori T, T' on a dimension δ $(1 \leq \delta \leq n)$, disjoint paths (at the exception that the paths for s_i and d_i $(1 \leq i \leq c)$ need not be disjoint) that route the vertices of one pair to T' without going through T and the vertices of the other pairs to T without going through T' can be found in $O(nc^2)$ time. Maximum path length is k.*

Proof. Assume that, given the pair (s_i, d_i) routed to T', there exists vertex s_j $(i \neq j)$ that cannot be disjointly routed to T without going through T', that is, each of all candidate paths to T for s_j as per Lemma 1 are blocked. See Figure 5. If such a vertex s_j does not exist, all vertices s_t, d_t $(1 \leq t \leq c, t \neq i)$ can be routed to T provided that if there exists a unique pair (s_j, d_j) $(j \neq i)$ with 3 disjoint candidate paths to T without going through T' blocked by the paths for (s_i, d_i) it is routed first to T, (i.e., before all the other pairs (s_t, d_t) $(1 \leq t \leq c, t \neq i, t \neq j)$), and there is thus nothing to prove.

For s_j to be fully blocked, vertex s_i or d_i $(i \neq j)$ needs to be on dimension δ, that is, blocking path $p(s_j, \delta)$. By definition of the candidate paths (see Lemma 1), this vertex s_i or d_i is the only one that can block two candidate paths for s_j, as all the other vertices s_l, d_l $(l \neq j, l \neq i)$ can block, at most, one candidate path for s_j. Hence, the $2(n-1)+1$ candidate paths for s_j are blocked by at least $2(n-1)$ vertices s_i or d_i $(i \neq j)$. In other words, for vertex s_j to be fully blocked, vertex s_i or d_i *needs* to be on $p(s_j, \delta)$, the unique direct path to T for s_j. Since at least $2(n-1)$ vertices s_i or d_i are required to fully block s_j, it means that one of these $2(n-1)$ blockers blocks $p(s_j, \delta)$ in addition to one of the two candidate paths for s_j on another dimension, say α $(\alpha \neq \delta)$. All other $2(n-1)-1$ blockers are positioned 2 per dimension β $(\beta \neq \delta, \beta \neq \alpha)$, at the exception of one blocker that is on the dimension α (on the other side of s_j compared with the blocker that blocks two paths at once for s_j). Hence, there exists one unique such vertex of $S \cup D$ (here s_j) that is not routable to T.

So, for a vertex s_j to be fully blocked, $T \cap (S \cup D) = \emptyset$ is induced, otherwise it would mean that at least one pair vertex needs not be routed at all (i.e., a path of zero length will do for that vertex), and, thus, all other vertices can be routed to T, T' by Lemma 1.

Furthermore, since $c \geq 3$, either there exists a pair (s_t, d_t) with $\{s_t, d_t\} \subset N(s_j)$ that can be routed to T' without going through T (in place of the pair (s_i, d_i)) with the paths $p(s_t, \delta)$ and $p(d_t, \delta)$ (see Figure 6a). Or, if d_j blocks one of these two paths, the pair (s_j, d_j) can be routed to T' without going through T (in place of the pair (s_i, d_i)) with the paths $p(s_j, \delta)$ and $p(d_j, \delta)$ (see Figure 6b). Because of the uniqueness of such a vertex s_j not routable to T, we have shown that it is possible to disjointly route one pair to T' and the other pairs to T.

Considering the two subtori T and T', the value of $|\gamma(T, \delta) - \gamma(T', \delta)|$ is at most $k - 1$. Given that the pair vertices are to be routed towards T (resp. T') not going through T' (resp. T), and by Lemma 1, the maximum path length is $(k - 2) + 2 = k$.

Regarding time complexity, pair vertices can be routed to T and T' as follows. Let (s_i, d_i) be the pair routed to T'. By Lemma 1, this takes $O(nc)$ time. If either s_i or d_i blocks a path $p(u, \delta)$ for u a vertex of a pair (s_j, d_j) $(1 \leq j \leq c, j \neq i)$, do as follows, and otherwise each of all pairs except the one routed to T' is routed to T with a path as per Lemma 1 that does not go through T'. Check if u is routable to T with a path as per Lemma 1 that does not go through T'; this takes $O(nc)$ time. If u is routable to T, starting with the pair (s_j, d_j), each of all pairs except (s_i, d_i) is routed to T with a path as per Lemma 1 that does not go through T'. If u not routable to T, discard the paths for (s_i, d_i) to T' and instead route an arbitrary pair (s_t, d_t) $(t \neq i, t \neq j)$ to T' with the paths $p(s_t, \delta), p(d_t, \delta)$ not going through T, with the possibility that d_j blocks (s_t, d_t) when routed to T', in which case it is the pair (s_j, d_j) that is routed to T', with the paths $p(s_j, \delta), p(d_j, \delta)$ not going through T. Each of all pairs except the one routed to T' is routed to T with a path as per Lemma 1 that does not go through T'. Hence, the induced total time complexity is that of Lemma 1: $O(nc^2)$. $\square$

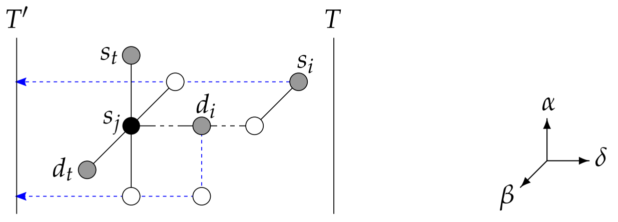

Figure 5. Illustration of the case $n = c = 3$ with the pair (s_i, d_i) blocking 3 paths for the pair (s_j, d_j) when routed to the subtorus T'. Vertex s_j is not routable to subtorus T without going through T'. (Dimension names are given on the right for reference.)

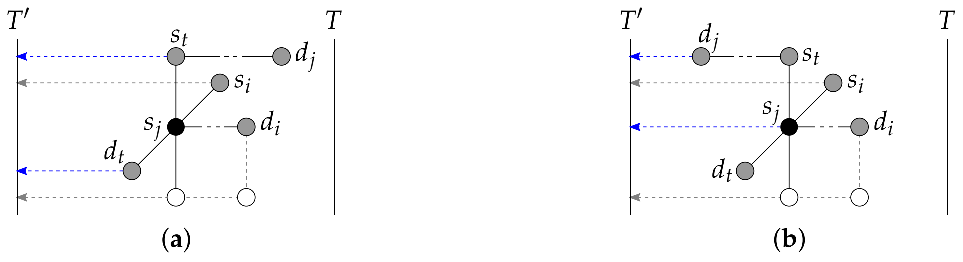

Figure 6. Illustration of the case $n = c = 3$ with the pair (s_i, d_i) blocking 3 paths for the pair (s_j, d_j) when routed to the subtorus T'. Vertex s_j is not routable to the subtorus T. (**a**) Pair (s_t, d_t) is routed to T' in place of (s_i, d_i). (**b**) Pair (s_t, d_t) is not routable to T' (because of d_j), so pair (s_j, d_j) is routed to T' in place of (s_i, d_i).

3. Routing Algorithm

In an (n, k)-torus $(n < k, k \geq 5)$, given c pairs (s_i, d_i) $(1 \leq i \leq c \leq n)$ satisfying $\{s_i, d_i\} \cap \{s_j, d_j\} = \emptyset$ $(1 \leq i, j \leq c, i \neq j)$; thus, $s_i = d_i$ allowed, this problem consists in finding a path connecting each (s_i, d_i) pair $(1 \leq i \leq c)$, and such that the selected c paths are mutually vertex-disjoint. The algorithm execution trace of a sample instance of the pairwise routing problem is given in Section 4.

First, one should note that a torus of dimension 1 (i.e., $n = 1$) is isomorphic to a ring. Hence, in this case, $c = 1$, and it is trivial to connect the unique source-destination vertex pair by traversing the ring. So, we henceforth assume that $n \geq 2$. The main idea of the proposed routing algorithm is to follow a divide-and-conquer approach by considering the (n, k)-torus as a k-set of $(n - 1, k)$-tori, connecting one source-destination pair in one such subtorus, and solving the problem recursively for the remaining pairs in another, distinct subtorus. We proceed according to the following cases.

3.1. Base Case: $c = 1$

This is point-to-point routing in a torus, and thus dimension-order routing can be applied to find a shortest path $s_1 \rightsquigarrow d_1$.

3.2. General Case: $n \geq 3$

Step 1 Find a subtorus T' such that the following three conditions hold:

$$\forall \gamma = \{s_i, d_j\} \in S \times D, i \neq j, \quad \gamma \not\subset T'$$
$$\forall \gamma' = \{s_i, s_j\} \in S^2, i \neq j, \quad \gamma' \not\subset T'$$
$$\forall \gamma'' = \{d_i, d_j\} \in D^2, i \neq j, \quad \gamma'' \not\subset T'$$

In other words, subtorus T' either includes no source vertex and no destination vertex, includes at most one source vertex and no destination vertex, includes at most one destination vertex and no source vertex, or includes one single source vertex, say s_i, and one single destination vertex d_i.

The selection of subtorus T' sets dimension δ to reduce the original (n, k)-torus to a set of k $(n-1, k)$-subtori.

Step 2 First, source-destination pair ρ is selected as follows. If $T' \cap (S \cup D) = \emptyset$, any pair will do, so select one arbitrarily, say $\rho = (s_i, d_i)$ $(1 \leq i \leq c)$. If $\exists i\, (1 \leq i \leq c), T' \cap (S \cup D) = \{s_i\}$, let $\rho = (s_i, d_i)$. Otherwise, that is $\exists i\, (1 \leq i \leq c), T' \cap (S \cup D) = \{d_i\}$, let $\rho = (s_i, d_i)$. In the remaining of this section, assume without loss of generality that $\rho = (s_i, d_i)$, in other words that the source-destination pair to be connected inside T' is (s_i, d_i).

Second, considering selected pair $\rho = (s_i, d_i)$, select subtorus T distinct from T', such that $\rho \cap T = \emptyset$.

Step 3 Route one source-destination pair to T' and the other pairs towards T as per Lemma 2.

Step 4 If for the pair (s_i, d_i) that is routed to T', say with the paths $p_s : s_i \rightsquigarrow s_i' \in T'$ and $p_d : d_i \rightsquigarrow d_i' \in T'$, we have $U \neq \emptyset$ with $U = (p_s \cap p_d) \setminus T'$, consider the vertex $u \in U$ that is the closest to s_i. Select the subpath $s_i \rightsquigarrow u$ of p_s and the subpath $d_i \rightsquigarrow u$ of p_d, and discard the subpath $u \rightsquigarrow s_i'$ of p_s and the subpath $u \rightsquigarrow d_i'$ of p_d. Otherwise, apply the algorithm recursively in T' to connect s_i to d_i.

Step 5 For each pair (s_j, d_j) routed to T, say with the paths $p_s : s_j \rightsquigarrow s_j' \in T$ and $p_d : d_j \rightsquigarrow d_j' \in T$, such that $U \neq \emptyset$ with $U = (p_s \cap p_d) \setminus T$, consider the vertex $u \in U$ that is the closest to s_j. Select the subpath $s_j \rightsquigarrow u$ of p_s and the subpath $d_j \rightsquigarrow u$ of p_d, and discard the subpath $u \rightsquigarrow s_j'$ of p_s and the subpath $u \rightsquigarrow d_j'$ of p_d. Define E the set of all such source-destination pairs.

Step 6 For each source-destination pair not in E, apply the algorithm recursively in T.

An illustration of the algorithm general case is given in Figure 7.

3.3. Special Case: $n = 2$ and $c = 2$

Select subtorus T' as in Step 1 of the general case, and deduce a pair ρ as in Step 2 of the general case. Route pair ρ to T' without going through T, and the other pair to T with at most one possible path going through T' (this may require exchanging the roles of T and T' as explained in the proof of correctness below).

For each of the two pairs (s_i, d_i) $(1 \leq i \leq 2)$, if the selected paths $p_s : s_i \rightsquigarrow s_i' \in T' \cup T$ and $p_d : d_i \rightsquigarrow d_i' \in T' \cup T$ are not disjoint, that is both include a same vertex u, discard the subpath $u \rightsquigarrow s_i'$ of p_s and the subpath $u \rightsquigarrow d_i'$ of p_d. Otherwise, complete the connection of s_i to d_i by traversing the subtorus of s'_i, d'_i. The subtori T', T are 1-dimensional and thus isomorphic to a ring. It is thus trivial to connect s'_i, d'_i, possibly avoiding one vertex.

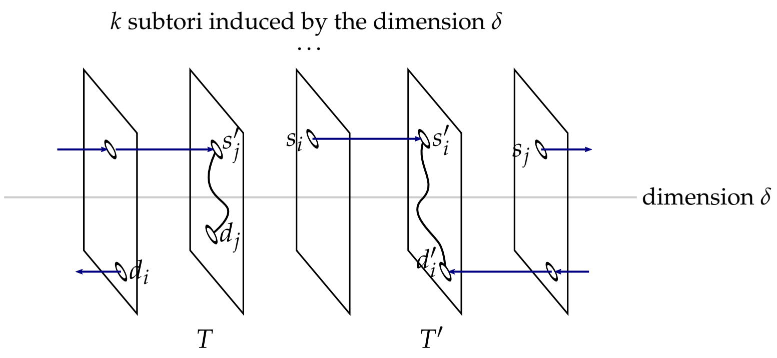

Figure 7. Disjointly connecting one source-destination pair inside subtorus T' and the other pairs inside subtorus T [9].

4. Execution Trace Example

In this section, considering the following disjoint pairwise routing problem instance, a possible execution trace of the proposed routing algorithm is detailed. In a (4, 5)-torus, let $S = \{s_1 = (2,1,0,4), s_2 = (0,2,1,2), s_3 = (2,4,0,2), s_4 = (4,4,4,1)\}$, $D = \{d_1 = (0,0,4,4), d_2 = (3,2,0,2), d_3 = (0,4,0,3), d_4 = (0,4,0,2)\}$ and $\{(s_1,d_1),(s_2,d_2),(s_3,d_3),(s_4,d_4)\}$ be the set of vertex pairs to be disjointly connected. Algorithm steps such as subtorus and subpath selection are given in Table 2. The torus arity, here $k = 5$, never changes and does thus not appear in the table.

Table 2. A possible algorithm execution trace in a (4, 5)-torus and the four vertex pairs $\{(s_1,d_1), (s_2,d_2),(s_3,d_3),(s_4,d_4)\}$ where $S = \{s_1 = (2,1,0,4), s_2 = (0,2,1,2), s_3 = (2,4,0,2), s_4 = (4,4,4,1)\}$ and $D = \{d_1 = (0,0,4,4), d_2 = (3,2,0,2), d_3 = (0,4,0,3), d_4 = (0,4,0,2)\}$.

n	Pairs	δ	$\gamma(T',\delta)$	$\gamma(T,\delta)$	Selected paths to T'
4	(s_1,d_1), (s_2,d_2), (s_3,d_3), (s_4,d_4)	1	1	3	$s_1 \to (1,1,0,4) = s'_1$
					$d_1 \to (1,0,4,4) = d'_1$
					Selected paths to T
					$s_4 \to (3,4,4,1) = s'_4$
					$d_4 \to (0,3,0,2) \to (4,3,0,2) \to (3,3,0,2) = d'_4$
					$s_3 \to (3,4,0,2) = s'_3$
					$d_3 \to (4,4,0,3) \to (3,4,0,3) = d'_3$
					$s_2 \to (4,2,1,2) \to (3,2,1,2) = s'_2$
					d_2
Path completion in T': $s'_1 \to (1,0,0,4) \to d'_1$.					
n	Pairs	δ	$\gamma(T',\delta)$	$\gamma(T,\delta)$	**Selected paths to T'**
3	(s'_2,d_2), (s'_3,d'_3), (s'_4,d'_4)	2	0	1	$s'_2 \to (3,3,1,2) \to (3,4,1,2) \to (3,0,1,2) = s''_2$
					$d_2 \to (3,2,4,2) \to (3,3,4,2) \to (3,4,4,2) \to (3,0,4,2) = d'_2$
					Selected paths to T
					$s'_3 \to (3,4,0,3) \to (3,3,0,3) \to (3,2,0,3) \to (3,1,0,3) = s''_3$
					$d'_3 \to (3,3,0,3) \to (3,2,0,3) \to (3,1,0,3) = d''_3$
					$s'_4 \to (3,3,4,1) \to (3,2,4,1) \to (3,1,4,1) = s''_4$
					$d'_4 \to (3,3,0,1) \to (3,2,0,1) \to (3,1,0,1) = d''_4$
Path completion in T': $s''_2 \to (3,0,0,2) \to d'_2$, and en route to T: $s'_3 \to d'_3$.					
n	Pairs	**Selected path**			
2	(s''_4,d''_4)	$s''_4 \to d''_4$ (base case $c = 1$: dimension-order routing)			

5. Proof of Correctness

First, dimension-order routing is applied in the base case of the recursion when there is one single source-destination pair. As recalled in Section 2, this process consists in traversing the torus one dimension after the other, concretely for each dimension δ starting from source vertex δ coordinate and traversing dimension δ until reaching destination vertex δ coordinate. This common algorithm is trivially correct.

For Step 1, it is required to show the existence of a subtorus T'. Along an arbitrary dimension δ, there are k ($k > n$) subtori T_i ($0 \leq i \leq k-1$). Amongst them, if there is a subtorus T_{i^*} that satisfies $|(S \cup D) \cap T_{i^*}| \leq 1$, we can select it as T'. Now, let us assume that such a subtorus T_{i^*} does not exist. Then, because $|(S \cup D) \cap T_i| \geq 2$ ($0 \leq i \leq k-1$), we have $\sum_{i=0}^{k-1} |(S \cup D) \cap T_i| \geq 2k > 2n$. This induces $|S \cup D| > 2n$, which is a contradiction. Hence, T_{i^*} always exists.

For Step 2, the existence of pair ρ is trivial. Regarding the existence of a subtorus $T \not\supset \rho$, it is recalled that the selection of T' fixes dimension δ inducing subtori. Therefore, on the one hand, excluding T', there remain $k-1$ candidate subtori for T. On the other hand, selected pair $\rho = (s_i, d_i)$ induces at most two additional unavailable subtori for T (i.e., if s_i, d_i in the same subtorus, only 1 additional unavailable subtorus is induced, and zero additional unavailable subtorus is induced if $\rho \subset T'$). Hence, there remain at least $k-3$ available subtori for T. Since $k \geq 5$, we have $k-3 \geq 2$.

The feasibility of Step 3 is proved by Lemma 2. For Step 4, at most one source-destination pair is to be connected recursively inside T'. Since $n \geq 3$, the dimension of T' is at least 2, and the problem can thus be solved recursively in T' (this is the base case $c = 1$ of the algorithm). In Steps 4 and 5, the two paths to the appropriate subtorus, say T, for a pair (s_j, d_j) are checked for the inclusion of a common vertex outside T. The only case for such a common vertex to exist is when the two paths are in the same direction (i.e., both taken either from P^+ or P^-), otherwise this would mean that the common vertex is in T. Hence, the common vertex outside T that is the closest to s_j is also the closest to d_j.

For Step 6, because one path is connected inside T' (or on the way to T'), the number of paths to select recursively in T is at most $c - 1 \leq n - 1$. Since T is of dimension $n-1$, the problem can be solved recursively inside T.

Finally, regarding the second base case of the recursion, the special case $c = 2, n = 2$, two source-destination pairs need to be connected inside a $(2, k)$-torus. Each of these two pairs is connected in a distinct subtorus. The existence of the two subtori T and T' has already been shown previously for the general case.

The vertices of pair ρ, say (s_1, d_1), are first routed to T' without going through T. Since $k \geq 5$, there always remains at least one path disjoint with the vertices (s_2, d_2) of the other pair to route (s_1, d_1) towards T'. For the second pair (s_2, d_2), the paths of (s_1, d_1) to T' may fully block (i.e., not routable as per Lemma 1 to T without going through T') one or both vertices of (s_2, d_2). If one single vertex of (s_2, d_2), say u, is fully blocked, route u towards T by going through T' with the path $q : p(u, \delta)$ to T through T'. This path q always remains unblocked since, for u to be fully blocked to T, one vertex of (s_1, d_1), say v, is on $p(u, \delta)$ to T, blocking two candidate paths to T for u, and the other vertex is in $N(u)$, precisely the neighbor of u that is opposite to the neighbor of u that is included in the path to T' for v. Hence, path $p(u, \delta)$ to T through T' is always disjoint with the paths to T' selected for (s_1, d_1). Since with the selection of $s_2, d_2 \notin T'$, and by the selection of the path q, subtorus T' includes one single vertex of the path q. See Figure 8a. If both vertices of (s_2, d_2) are fully blocked by the paths to T' for (s_1, d_1), it means that $(T \cup T') \cap (S \cup D) = \emptyset$ and that s_2, d_2 are blocking both $p(s_1)$ and $p(d_1)$ towards T. Hence, route instead (s_2, d_2) to T' (with $p(s_2, \delta), p(d_2, \delta)$ to T') and (s_1, d_1) to T (with $p(s_1, \delta), p(d_1, \delta)$ to T). See Figure 8b.

The two subtori T, T' are one-dimensional, that is, isomorphic to a ring. It is thus trivial to complete the connection of each of the two pairs inside their respective subtori, even if there is one vertex to be avoided in subtorus T': either the clockwise or counter-clockwise ring traversal will do.

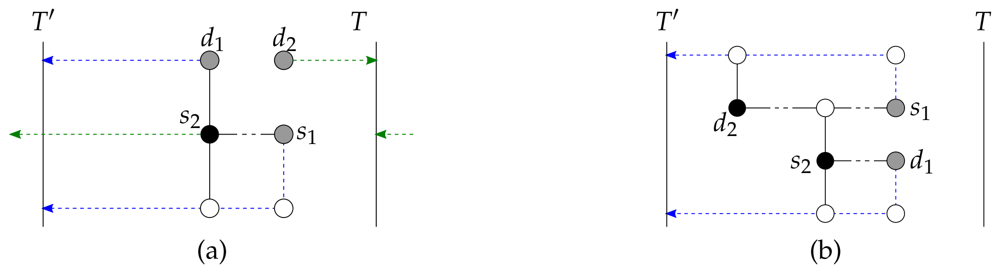

Figure 8. Illustration of the case $n = c = 2$ with the pair (s_1, d_1) originally selected to be routed to T'. (**a**) The vertex s_2 is fully blocked to T if not going through T'. (**b**) Both s_2, d_2 are fully blocked to T: (s_2, d_2) is routed instead to T', and (s_1, d_1) to T.

6. Complexities

Let function $T(c, n)$ represent the time complexity of the proposed algorithm in an (n, k)-torus with c source-destination pairs ($c \leq n$). And, let the function $L(c, n)$ represent the maximum length of a path selected by the algorithm inside an (n, k)-torus with c source-destination pairs ($c \leq n$).

The base case $c = 1$ of the recursion induces the selection of a path with a dimension-order routing algorithm, thus inducing an $O(nk)$ time complexity and a maximum path length of $n\lfloor k/2 \rfloor$. Hence, $T(1, n) = O(nk)$ and $L(1, n) = n\lfloor k/2 \rfloor$.

For Step 1, subtorus T' is selected. A subtorus is selectable as T' if the stated three conditions hold. The first condition can be checked by iterating the source and destination vertices, testing inclusion inside the current subtorus. Testing vertex inclusion in a subtorus is achieved by checking the vertex coordinate for each dimension, inducing then an $O(n)$ time complexity. Hence, the first condition is checked for one subtorus in $O(nc^2)$ time. Checking the second and third conditions induces the same time complexity. Therefore, since for an arbitrary dimension δ at most c subtori are unavailable as T', it is needed to check at most c subtori for the three conditions, thus inducing an $O(nc^3)$ total time complexity.

For Step 2, selecting the pair ρ is done by iterating each source-destination pair, each time testing the inclusion of the pair source and destination vertices inside T'. A total time complexity of $O(cn)$ is thus induced. The time complexity of Step 3 is directly induced by Lemma 2: $O(nc^2)$. For Step 4, the two paths to T' are checked similarly, which thus induces an $O(nk)$ time complexity. In addition, if these two paths do not collide, the pair connection is completed in an $(n-1, k)$-torus, thus inducing an $O(nk)$ time complexity. For Step 5, the two paths to T for each pair are checked for inclusion of a common vertex. The paths for one pair are thus checked in $O(nk)$ time. Hence, all pair paths are checked in $O(cnk)$ time. For Step 6, time complexity is $T(c-1, n-1)$.

In the special case $n = 2, c = 2$, two subtori T, T' are selected as in Step 1 of the general case, thus inducing an $O(1)$ time complexity. The selection of the paths for the two pairs may require checking all the candidate paths as per Lemma 1, thus inducing an $O(n)$ time complexity. An additional $O(nk)$ is required to check whether the selected pair paths are disjoint. Finally, routing inside one-dimensional subtori is possible, checking whether the selected path includes the vertex to avoid ($O(1)$), thus being $O(k)$ time. Hence, in total, this special case is $O(nk)$ time. The maximum path length is obtained when k links are taken to route both the source and destination vertices to the designated subtorus, and with $k-1$ links for routing inside one-dimensional subtori, thus in total $3k-1$.

The above discussion can be summarized in the following theorem.

Theorem 1. *In an (n, k)-torus ($n < k, k \geq 5$), given c ($1 \leq c \leq n$) vertex pairs (s_i, d_i) (all pair vertices are distinct, yet $s_i = d_i$ is acceptable), it is possible to select c mutually vertex-disjoint paths $s_i \rightsquigarrow d_i$ ($1 \leq i \leq c$) of lengths at most $2k(c-1) + n\lfloor k/2 \rfloor$ in $O(nc^4)$ time.*

Proof. A path of maximum length could be selected as follows: each recursive step induces k links to route the source vertex to the designated subtorus, k links to route the destination vertex to the designated subtorus, and this until reaching a base case, either $c = 1$ with a path of $n\lfloor k/2 \rfloor$ links selected by dimension-order routing, or $n = 2, c = 2$ with two paths each of length at most $3k - 1$. This is summarized with the following recursive expression:

$$\begin{aligned} L(1,n) &= n\lfloor k/2 \rfloor. \\ L(2,2) &= 3k - 1. \\ L(c,n) &= 2k + L(c-1, n-1). \end{aligned}$$

Hence, the maximum path length is $\max\{2k(c-1) + n\lfloor k/2 \rfloor, 2k(c-2) + (3k-1)\}$, which is equal to $2k(c-2) + 3k - 1$ if $n = 1$ and to $2k(c-1) + n\lfloor k/2 \rfloor$ otherwise, given that $k \geq 5$. However, since in the $n = 1$ case we have, on the one hand, $c = 1$ and, on the other hand, $L(1,1) = \lfloor k/2 \rfloor$, the maximum path length can be expressed as $2k(c-1) + n\lfloor k/2 \rfloor$ for any n.

The total worst-case time complexity is given by the following recursive expression:

$$\begin{aligned} T(1,n) &= O(nk). \\ T(2,2) &= O(nk). \\ T(c,n) &= O(nc^3) + T(c-1, n-1). \end{aligned}$$

Hence, the total time complexity of the proposed routing algorithm is $O(nc^4)$. □

7. Empirical Evaluation

We have implemented the algorithm proposed in this paper in order to realize its empirical evaluation, and especially to inspect its practical behavior: how the algorithm performs in average. The computer used for this experiment was equipped with an Intel Core i5-7300U CPU (clocked at 2.60 GHz), 8 GB RAM and ran Windows 10 64-bit.

By using this computer implementation, we have repetitively and automatically solved a large number of random instances of the pairwise disjoint-path routing problem in a torus. Precisely, in an $(n, \max\{5, n+1\})$-torus, we have solved for each value of n ($2 \leq n \leq 7$) a total of 10,000 random routing problem instances. For each of these instances, the correctness of the selected paths has been checked (e.g., path disjointness), and the maximum length of the selected paths was stored. Then, for each value of n, the average of the stored maximum path lengths for this value of n was computed. Hence, our results include for each n both the maximum and the average of the obtained 10,000 maximum path lengths. The number of vertex pairs was set to $c = n$ to maximize the routing difficulty. For each n, the maximum and average of the maximum path lengths are given in Figure 9; the theoretical upper bound on the maximum path length is also given for reference.

Then, we have measured the average execution time for one problem instance: see Figure 10; the worst-case time complexity is also plotted for reference.

Regarding maximum path lengths, one can first note that there is some distance between the theoretical upper bound on the maximum path length and the obtained results. This is a good indicator of the efficiency of the proposed algorithm, especially given that for small values of n the upper bound is almost tight on the maximum path length. Regarding time complexity, it can be observed from the obtained results that the average performance of the algorithm is significantly better than the theoretical worst-case time complexity.

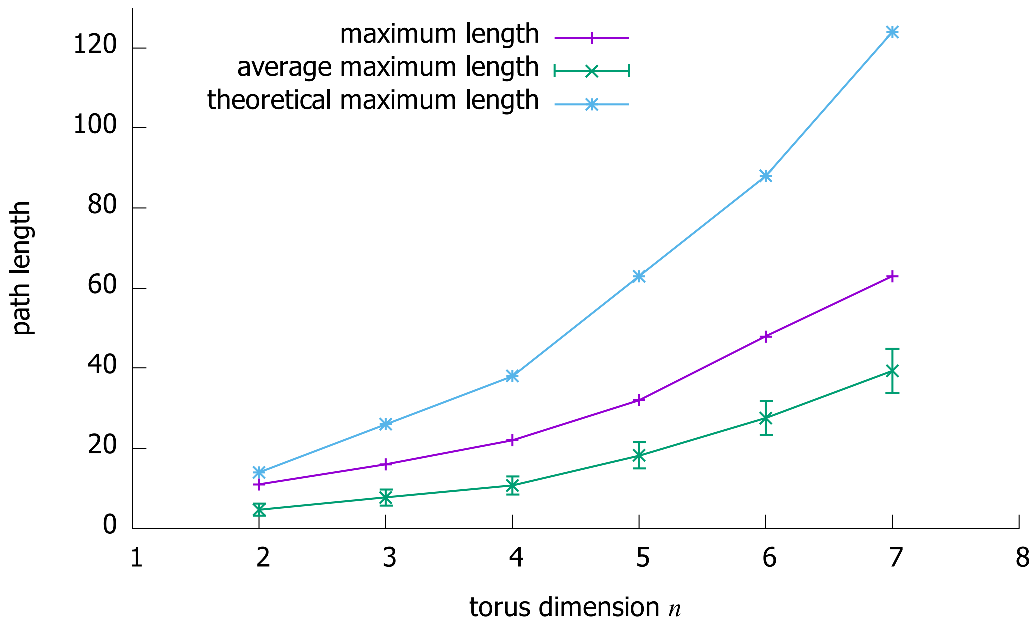

Figure 9. Empirical evaluation: maximum and average maximum (with standard deviation) path lengths of paths selected by the proposed algorithm in an $(n, \max\{5, n+1\})$-torus.

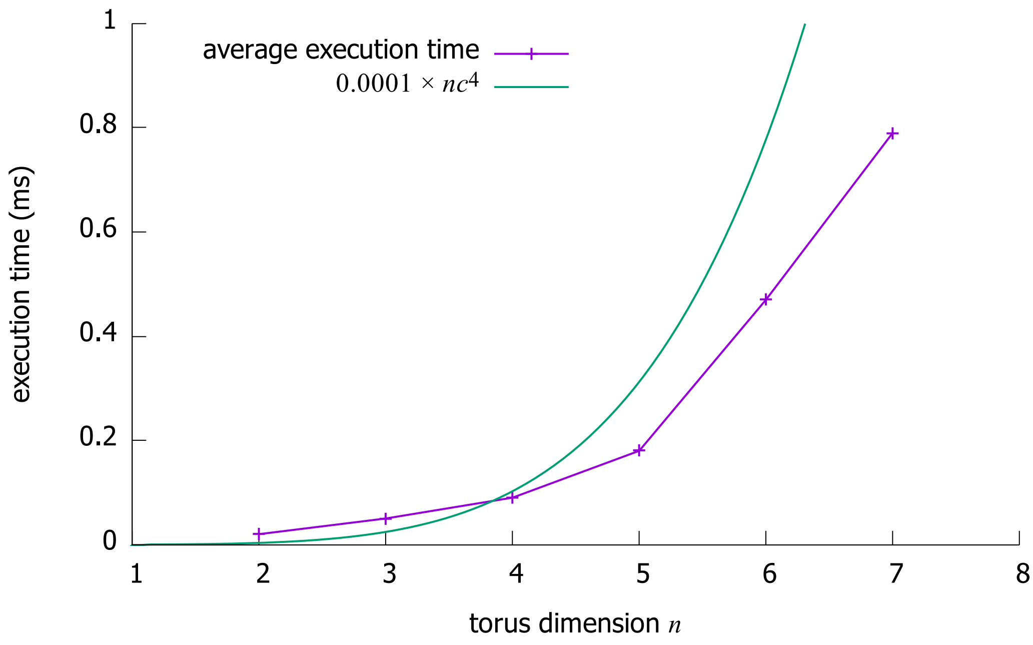

Figure 10. Empirical evaluation: average execution time to solve one problem instance with the proposed algorithm in an $(n, \max\{5, n+1\})$-torus.

8. Conclusions

The torus topology is very popular for the interconnection network of massively parallel systems such as the modern supercomputers Fujitsu K, IBM Blue Gene/L, IBM Blue Gene/P, and Cray Titan. In this paper, an algorithm that solves the pairwise disjoint-path routing problem in a torus has been proposed. This routing problem consists in the selection of mutually vertex-disjoint paths between given vertex pairs, and it is critical to maximize system dependability and data communication efficiency. We have then formally shown that in an n-dimensional k-ary torus, for c ($c \leq n$) vertex pairs, the described algorithm selects disjoint paths of lengths at most $2k(c-1)+n\lfloor k/2 \rfloor$ with a worst-case time complexity of $O(nc^4)$. Furthermore, the practical behavior of the proposed algorithm has been inspected by conducting computer experiments, showing that performance on average was significantly better than the established formal worst-case complexities.

As for future works, it will be meaningful to investigate whether the theoretical maximum path length is attainable, and if not, try to refine it. For example, this might be achieved by more wisely selecting the subtori used for disjoint routing. Even though the selection of subtorus T' may not leave many options, the selection of subtorus T that was used to solve the problem recursively could be done such that shorter paths to route pair vertices to T are selected.

Author Contributions: Conceptualization, A.B. and K.K.; Formal analysis, A.B. and K.K.; Investigation, A.B. and K.K.; Methodology, A.B. and K.K.; Software, A.B.

Funding: This research was partly supported by a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science under Grant no. 17K00093.

Acknowledgments: The authors sincerely thank the reviewers for their insightful comments and suggestions.

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

Appendix A. Additional Details of Candidate Paths

In an (n, k)-torus, given vertex $u = (u_1, u_2, \ldots, u_n)$ and dimension δ $(1 \le \delta \le n)$, path $p^-(u,\delta)$ and path sets $P_1^-(u,\delta)$, $P_2^-(u,\delta)$ are defined similarly to $p^+(u,\delta)$, $P_1^+(u,\delta)$, and $P_2^+(u,\delta)$, respectively, as described in Section 2. The formal definitions of the path $p^-(u,\delta)$ and the path sets $P_1^-(u,\delta)$, $P_2^-(u,\delta)$ are given below.

$$p^-(u,\delta): \quad u \to u \ominus e_\delta \to (u \ominus e_\delta) \ominus e_\delta \to \ldots$$

$$P_1^-(u,\delta): \quad \bigcup_{\substack{1\le i\le n\\ i\ne\delta}} \left\{ \begin{array}{l} u \to u \oplus e_i \to (u \oplus e_i) \ominus e_\delta \to ((u \oplus e_i) \ominus e_\delta) \ominus e_\delta \to \ldots, \\ u \to u \ominus e_i \to (u \ominus e_i) \ominus e_\delta \to ((u \ominus e_i) \ominus e_\delta) \ominus e_\delta \to \ldots \end{array} \right\}$$

$$P_2^-(u,\delta): \quad \bigcup_{\substack{1\le i\le n\\ i\ne\delta}} \left\{ \begin{array}{l} u \to u \oplus e_i \to (u \oplus e_i) \oplus e_i \to ((u \oplus e_i) \oplus e_i) \ominus e_\delta \\ \quad \to (((u \oplus e_i) \oplus e_i) \ominus e_\delta) \ominus e_\delta \to \ldots, \\ u \to u \ominus e_i \to (u \ominus e_i) \ominus e_i \to ((u \ominus e_i) \ominus e_i) \ominus e_\delta \\ \quad \to (((u \ominus e_i) \ominus e_i) \ominus e_\delta) \ominus e_\delta \to \ldots \end{array} \right\}$$

References

1. Seitz, C.L. The cosmic cube. *Commun. ACM* **1985**, *28*, 22–33. [CrossRef]
2. Meritt, R. Cray Studies Exascale Computing in Europe. Available online: http://www.eetimes.com/document.asp?doc_id=1268565 (accessed on 27 September 2018).
3. TOP500 Team. TOP500 list refreshed, US Edged Out of Third Place. 2017. Available online: https://www.top500.org/news/top500-list-refreshed-us-edged-out-of-third-place/ (accessed on 27 September 2018).
4. Li, Y.; Peng, S.; Chu, W. Efficient collective communications in dual-cube. *J. Supercomput.* **2004**, *28*, 71–90. [CrossRef]
5. Li, Y.; Peng, S.; Chu, W. Metacube—A versatile family of interconnection networks for extremely large-scale supercomputers. *J. Supercomput.* **2010**, *53*, 329–351. [CrossRef]
6. Akers, S.; Krishnamurthy, B. A group-theoretic model for symmetric interconnection networks. *IEEE Trans. Comput.* **1989**, *C-38*, 555–566. [CrossRef]
7. Gates, W.; Papadimitriou, C. Bounds for sorting by prefix reversal. *Discret. Math.* **1979**, *27*, 47–57. [CrossRef]
8. Duato, J.; Yalamanchili, S.; Ni, L. *Interconnection Networks: An Engineering Approach*; Morgan Kaufmann: Burlington, MA, USA, 2003.
9. Bossard, A.; Kaneko, K. On the torus pairwise disjoint-path routing problem. In Proceedings of the 18th IEEE International Conference on Computer and Information Technology (CIT), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1739–1746.

10. Ajima, Y.; Inoue, T.; Hiramoto, S.; Takagi, Y.; Shimizu, T. The Tofu interconnect. *IEEE Micro* **2012**, *32*, 21–31. [CrossRef]
11. Ajima, Y.; Inoue, T.; Hiramoto, S.; Uno, S.; Sumimoto, S.; Miura, K.; Shida, N.; Kawashima, T.; Okamoto, T.; Moriyama, O.; et al. Tofu interconnect 2: System-on-chip integration of high-performance interconnect. In Proceedings of the 29th International Supercomputing Conference, Leipzig, Germany, 22–26 June 2014; pp. 498–507.
12. Cray Inc. Cray XE6 Brochure. 2010. Available online: http://www.cray.com/products/computing/xe-series (accessed on 27 September 2018).
13. Murugesan, S. Harnessing green IT: Principles and practices. *IT Prof.* **2008**, *10*, 24–33. [CrossRef]
14. Green500. June 2017 list. Available online: https://www.top500.org/green500/ (accessed on 27 September 2018).
15. Menger, K. Zur allgemeinen Kurventheorie. *Fund. Math.* **1927**, *10*, 96–115. [CrossRef]
16. Karp, R.M. On the complexity of combinatorial problems. *Networks* **1975**, *5*, 45–68. [CrossRef]
17. Robertson, N.; Seymour, P.D. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory Ser. B* **1995**, *63*, 65–110. [CrossRef]
18. Jerebic, I.; Trobec, R. Optimal routing in toroidal networks. *Inf. Process. Lett.* **1992**, *43*, 285–291. [CrossRef]
19. Gu, Q.-P.; Peng, S. Fault tolerant routing in toroidal networks. *IEICE Trans. Inf. Syst.* **1996**, *E79-D*, 162–168.
20. Kaneko, K.; Bossard, A. A set-to-set disjoint paths routing algorithm in tori. *Int. J. Netw. Comput.* **2017**, *7*, 173–186. [CrossRef]
21. Gu, Q.-P.; Peng, S. An efficient algorithm for the *k*-pairwise disjoint paths problem in hypercubes. *J. Parallel Distrib. Comput.* **2000**, *60*, 764–774. [CrossRef]
22. Gu, Q.-P.; Peng, S. An efficient algorithm for *k*-pairwise disjoint paths in star graphs. *Inf. Process. Lett.* **1998**, *67*, 283–287. [CrossRef]
23. Bossard, A.; Kaneko, K. *k*-pairwise disjoint paths routing in perfect hierarchical hypercubes. *J. Supercomput.* **2014**, *67*, 485–495. [CrossRef]
24. Sawada, N.; Kaneko, K.; Peng, S. Pairwise disjoint paths in pancake graphs. In Proceedings of the 8th International Conference on Parallel and Distributed Computing, Applications and Technologies, Adelaide, Australia, 3–6 December 2007; pp. 376–382.
25. Park, J.-H. Paired many-to-many disjoint path covers in restricted hypercube-like graphs. *Theor. Comput. Sci.* **2016**, *634*, 24–34. [CrossRef]
26. Wang, X.; Fan, J.; Jia, X.; Lin, C.-K. An efficient algorithm to construct disjoint path covers of DCell networks. *Theor. Comput. Sci.* **2016**, *609*, 197–210. [CrossRef]
27. Cheng, B.; Fan, J.; Jia, X.; Jia, J. Parallel construction of independent spanning trees and an application in diagnosis on Möbius cubes. *J. Supercomput.* **2013**, *65*, 1279–1301. [CrossRef]
28. Liu, J.; Zhao, J.; Wang, S.; Javaid, M.; Cao, J. On the topological properties of the certain neural networks. *J. Artif. Intell. Soft Comput. Res.* **2018**, *8*, 257–268. [CrossRef]
29. Wang, C.; Yu, G.; Sun, W.; Cao, J. The least eigenvalue of the graphs whose complements are connected and have pendent paths. *J. Artif. Intell. Soft Comput. Res.* **2018**, *8*, 303–308. [CrossRef]