International Journal of Networked and Distributed Computing, Vol. 3, No. 4 (November 2015), 202-213

A Routing Algorithm Solving the Container Problem in a Hypercube with Bit Constraint

Antoine Bossard [1], Keiichi Kaneko [2]

[1] *Graduate School of Science, Kanagawa University,*
2946 Tsuchiya, Hiratsuka,
Kanagawa, Japan 259-1293
E-mail: abossard@kanagawa-u.ac.jp

[2] *Graduate School of Engineering, Tokyo University of Agriculture and Technology,*
Nakacho 2-24-16, Koganei,
Tokyo, Japan 184-8588
E-mail: k1kaneko@cc.tuat.ac.jp

Abstract

As reflected by the TOP500 list, hypercubes are popular interconnection networks for massively parallel systems, the main reason being the simplicity and ease of implementation of this network topology. In order to retain performance high and avoid bottleneck situation, routing algorithms are critical for these high-performance systems. Furthermore, *disjoint* path routing is a very desirable property of such communication algorithms. Effectively, selecting mutually node-disjoint paths guarantees that notorious parallel processing issues such as deadlocks, livelocks and starvations shall never occur. In this paper, we describe a routing algorithm for hypercubes that, given a bit constraint, selects internally node-disjoint paths between any pair of nodes satisfying the constraint, and such that the selected paths all satisfy the constraint. The introduction of such bit constraint enables the selection of multiple sets of disjoint paths between several node pairs each satisfying a distinct bit constraint, which is impossible with conventional routing algorithms. Selecting simultaneously disjoint paths between different node pairs induces increased communication performance and system dependability. The correctness and complexities of the described algorithm are formally proved, and analysis of the algorithm performance in practice is conducted by empirical evaluation.

Keywords: Supercomputer, parallel system, network, disjoint paths, dependable system, node-to-node.

1. Introduction

Thanks to a simple topology definition, and thus facilitated hardware and software implementation, hypercubes[1] are popular interconnection networks for massively parallel systems. Such hypercube-based machines feature a decades long history[2], with very recent ones including the NASA Pleiades and NOAA Zeus supercomputers[3]. Additionally, one should note that hypercubes are very popular as seed (i.e. sub-network) of more complex interconnection network topologies, especially those for hierarchical interconnection networks (HINs) with examples such as dual-cubes[4], metacubes[5], hierarchical hypercubes[6] and hierarchical cubic networks[7].

It thus easy to understand that because of these reasons, routing in hypercubes is a critical and actively researched topic. Several routing algorithms for hypercubes have been described in the literature. For example, optimal node-to-node disjoint

paths routing algorithm[8,9], node-to-set disjoint paths routing algorithm[10], set-to-set disjoint paths routing algorithm[11] and k-pairwise disjoint paths routing algorithm[12]. Now, these conventional approaches to hypercube routing do not enable enforcing a constraint on the nodes selected when generating paths. Yet, routing with constraint has interesting properties, allows for new applications, and importantly induces increased performance and system dependability. Indeed, by enforcing restriction on selected nodes, it is easy to achieve disjoint paths routing between several node pairs, at the condition that each pair satisfies a distinct constraint. This topic has been discussed previously[13], and is actually a convenient method to achieve path signature as presented in[14].

The selection of node-disjoint paths is a very desirable property of routing algorithms in parallel systems. Effectively, communication conducted according to mutually disjoint paths ensures the absence of notorious resource allocation problems of parallel systems, namely deadlocks, livelocks and starvations. Furthermore, as parallel communication enables more efficient utilisation of the interconnection network, simultaneous path selection has a critical positive impact on system performance, and with implications going as far as Green IT[15]. In addition, dramatically increased system dependability is yet another critical aspect of disjoint paths routing. Indeed, with modern supercomputers now including a huge number of computing nodes (e.g. 705,024 in the Fujitsu K[3]), there is a high probability that faulty nodes will be present[16]. Here, by enforcing selection of mutually node-disjoint paths, the impact of one such broken node is severely limited: one fault can neutralise at most one path thanks to the mutual disjointness of the selected paths.

In this paper, we describe a routing algorithm that solves the container problem with bit constraint in a hypercube. This routing problem is also known as the node-to-node disjoint paths routing problem, and consists of finding a set of mutually internally node-disjoint paths between any pair of nodes[17,18,19]. And by considering routing with bit constraint, we further increase the advantages of disjoint paths routing. Concretely, by considering several node pairs each satisfying a distinct bit constraint, we make it possible to easily select *at the same time* several sets of disjoint paths between these node pairs, something which is impossible with conventional algorithms, even with disjoint paths routing algorithms. This added routing capability enables selecting even more disjoint paths that are available simultaneously for routing operations, and thus significantly improves communication performance and system dependability. We give an illustration in Figure 1.

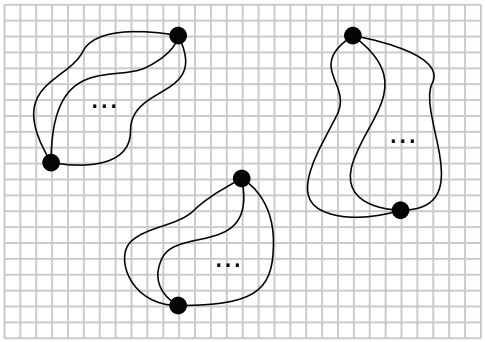

Figure 1: Several sets of node-to-node disjoint paths, enabling high performance network communications and increases system dependability.

The rest of this paper is organised as follows. First, the notations and definitions used in the paper hereinafter are recalled in Section 2. Then, we describe in Section 3.1 the hypercube node-to-node disjoint paths routing algorithm with bit constraint, and we give the corresponding pseudo-code. Next, the algorithm correctness and complexities are formally proved in Section 3.2. This is followed in Section 3.3 by an example of the algorithm execution trace. Then, an empirical evaluation of the described algorithm is conducted so as to inspect its practical behaviour and to make a comparison with the established theoretical results of the previous sections. Lastly, we conclude this paper in Section 5.

2. Preliminaries

We recall in this section several definitions, notations and results used throughout the paper. In addition, new notations are introduced.

Definition 1. An n-dimensional hypercube, denoted by Q_n, consists of 2^n nodes, each having a unique n-bit address. Two nodes u and v of a hypercube are adjacent if and only if their Hamming distance $H(u,v)$ is equal to one.

Let us recall important topological properties of hypercubes. First, a Q_n is symmetric and of connectivity, degree and diameter n (see[1]). In addition, one should remember that a Q_n has a recursive structure. Effectively, for any dimension δ ($0 \leqslant \delta \leqslant n-1$), a Q_n consists of two $(n-1)$-dimensional hypercubes Q^0_{n-1} and Q^1_{n-1}, here called subcubes, and defined as follows. The subcube Q^0_{n-1} (resp. Q^1_{n-1}) is induced by the set of nodes of Q_n whose δ-th bits are set to 0 (resp. 1). When considering the subcubes of a hypercube, we talk of *hypercube reduction*. As an example, a 4-dimensional hypercube Q_4 with its two subcubes Q^0_3 and Q^1_3 highlighted is illustrated in Figure 2.

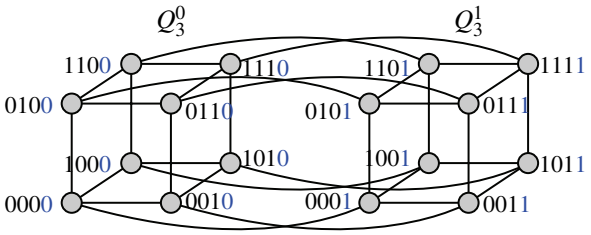

Figure 2: A 4-dimensional hypercube Q_4 with its two subcubes Q^0_3 and Q^1_3 induced by $\delta = 0$.

We assume that the address of any node of a Q_n (i.e. an n-bit address) can be stored in a fixed number of machine words, therefore allowing for constant time node comparison, most significant bit (MSB) detection as well as Hamming distance and bit weight (a.k.a. Hamming weight, see Definition 2 below) calculations.

Definition 2. For a binary n-bit sequence $b = b_{n-1}\ldots b_1 b_0$, $b_i \in \{0,1\}$, $0 \leqslant i \leqslant n-1$, the *bit weight* of b, denoted by $w(b)$, is the number of bits of b that are set to 1.

Also, let us adopt the following conventions: in this paper, logarithms are in base two, and the MSB of any bit sequence is the leftmost bit. In addition, the binary bitwise operations are denoted as follows: the bitwise AND is denoted by &, the binary negation is denoted by $\neg$, and the bitwise exclusive-OR is denoted by $\oplus$.

Definition 3. A k-constraint is a k-tuple of distinct natural numbers $(i_1, i_2, \ldots, i_k)$.

In this paper, we apply such constraint to the bit weight of the address of a hypercube node. We focus on 2-constraints and simply speak of *bit constraints*, which are denoted by pairs of natural numbers (i, j). Since we are considering routing inside hypercubes where we recall that adjacent nodes have one single bit different, it is easy to understand that bit constraints considered all have the form $(i, i+1)$. And if we were to consider k-constraints on hypercubes, those bit constraints would have the form $(i, i+1, \ldots, i+\beta)$ with $i+\beta \leqslant n$.

Definition 4. In a Q_n, for $i \in \mathbb{N}$ and $0 \leqslant i \leqslant n-1$, a node u satisfies the constraint $\gamma_i = (i, i+1)$ if and only if $w(u) = i$ or $w(u) = i+1$ holds.

For example, let us consider a hypercube Q_3 and the bit constraint $\gamma_1 = (1,2)$. So, the three nodes 010, 110 and 100 all satisfy the constraint γ_1 whereas the node 111 does not.

Now, we recall definitions and notations that concern paths. In any graph, a path is an alternate sequence of nodes and edges. For a path p, we write $p : u_1, (u_1,u_2), u_2, \ldots, u_{k-1}, (u_{k-1},u_k), u_k$, with (u_i, u_{i+1}) denoting the edge between the two distinct nodes u_i and u_{i+1}. Conveniently, that same path p can also be written as $u_1 \to u_2 \to \ldots \to u_k$, and even more concisely as $u_1 \rightsquigarrow u_k$, the latter notation possibly bringing ambiguity regarding the nodes included in the path and thus a notation to be used with care. The length of a path corresponds to the number of its edges.

Two paths are mutually node-disjoint (or simply *disjoint*) if and only if they have no node in common. Conveniently, we consider a path as a set of nodes, and thus the two paths p_1 and p_2 are disjoint if and only if $p_1 \cap p_2 = \emptyset$ holds. We say that two paths are *internally disjoint* if and only if they have no node in common at the possible exception of their terminal nodes (i.e. the two nodes that start and end the path). Formally, two paths $p_1 : u_1 \rightsquigarrow v_1$ and $p_2 : u_2 \rightsquigarrow v_2$ are internally disjoint if and only if $p_1 \cap p_2 \subseteq \{u_1, u_2, v_1, v_2\}$ holds. We recall that

the container problem (a.k.a. the node-to-node disjoint paths routing problem) consists in selecting internally disjoint paths between any pair of distinct nodes.

Definition 5. A path p connecting a node u to a node v satisfies the constraint $\gamma_i = (i, i+1)$ if and only if each node of p satisfies γ_i. We write $u \overset{\gamma_i}{\rightsquigarrow} v$, or simply $u \overset{\gamma}{\rightsquigarrow} v$.

Hence, since the Hamming distance between any two adjacent nodes in a hypercube is equal to one, a path cannot satisfy a 2-constraint other than that of the form $(i, i+1)$ (or $(i, i-1)$, which is equivalent). If the path $u \overset{\gamma}{\rightsquigarrow} v$ connecting the two nodes u and v while satisfying a constraint γ_i has been generated by a shortest-path routing algorithm, we write $u \overset{\gamma,\,\mathrm{spr}}{\rightsquigarrow} v$ to indicate that it is a shortest path.

Finally, let us recall in the following theorem a previous result regarding hypercube shortest-path routing with bit constraint[13].

Theorem 1. [13] *In a Q_n, given a bit constraint $\gamma_i = (i, i+1)$ and any two distinct nodes s and d that satisfy γ_i, we can select a shortest path $s \overset{\gamma}{\rightsquigarrow} d$ (i.e. of length $H(s,d)$) satisfying γ_i in $O(H(s,d))$ optimal time.*

This algorithm is referred to as HC-SPR thereafter.

3. Node-to-node disjoint paths routing algorithm with γ_i constraint

First, given a node $u \in Q_n$ satisfying $\gamma_i = (i, i+1)$, we begin this section by discussing the number of its neighbours that satisfy γ_i. If $w(u) = i+1$, then u has $i+1$ neighbours satisfying the constraint. If $w(u) = i$, then u has $n-i$ neighbours satisfying the constraint. Therefore, in a Q_n, given two nodes s, d satisfying γ_i, we can select at most $k \leqslant \min(n-i, i+1)$ internally node-disjoint paths $s \overset{\gamma}{\rightsquigarrow} d$ that satisfy γ_i (this is an application of Menger's theorem[20]). Also, one can note that in the case $i = 0$, the maximum number of disjoint paths that can be selected is $\min(n-i, i+1) = 1$, and it is thus more efficient to apply the shortest-path routing algorithm of Theorem 1. So, let us assume without loss of generality that $i \geqslant 1$. The proposed algorithm pseudo-code is given in Algorithm 1 with sub-cases in Algorithms 2 and 3.

Algorithm 1 HC-CONTAINER(Q_n, i, k, s, d)

Input: A Q_n, a bit constraint $\gamma_i = (i, i+1)$, k the number of paths to find ($k \leqslant \min(n-i, i+1)$), a source node s and a destination node d.

Output: k internally node-disjoint paths $s \overset{\gamma}{\rightsquigarrow} d$ in Q_n satisfying γ_i.

```
1: if k = 1 then
2:     return HC-SPR(Q_n, i, s, d)
3: else if w(s) = i then
4:     return CASE1(Q_n, i, k, s, d)
5: else // w(s) = i + 1
6:     return CASE2(Q_n, i, k, s, d)
```

3.1. Algorithm description

If $n - i = 0$, the constraint $\gamma_i = (i, i+1)$ cannot be satisfied since $i + 1 > n$. If $n - i = 1$, the constraint $\gamma_i = (i, i+1)$ implies that only the nodes of weights n and $n-1$ can be selected, hence $H(s,d) \leqslant 2$. In this special case, the problem is solved as follows. If $H(s,d) = 1$, there exists only one path $s \overset{\gamma}{\rightsquigarrow} d$: the path of length one $s \overset{\gamma,\,\mathrm{spr}}{\rightsquigarrow} d = s \to d$. If $H(s,d) = 2$, there exists only one path $s \overset{\gamma}{\rightsquigarrow} d$: the path of length two $s \overset{\gamma,\,\mathrm{spr}}{\rightsquigarrow} d = s \to u \to d$ with u the unique node of Q_n of weight n. So, we can now assume without loss of generality that $n - i \geqslant 2$.

The main idea of this algorithm is to follow a divide-and-conquer approach by solving the problem recursively in one of the two subcubes Q^0_{n-1} and Q^1_{n-1} of the original network Q_n. The base case of this induction process is $k = 1$, with $k \leqslant \min(n-i, i+1)$ the number of paths to find, decremented at each recursive call. This base case $k = 1$ induces either $i = 0$ or $i = n-1$, and each of these two cases induces the selection of one single path (shortest) as already discussed. We distinguish two cases.

3.1.1. Case 1: $w(s) = i$

We proceed in several main steps as follows. Pseudo-code is given in Algorithm 2.

Step 1 Find a bit position δ ($0 \leqslant \delta \leqslant n-1$) such that the δ-th bit of s is set to 0 and the δ-th bit of d is set to 1.

Reducing the hypercube Q_n along this bit position δ, we obtain the two subcubes Q^0_{n-1} and Q^1_{n-1}, and $s \in Q^0_{n-1}$, $d \in Q^1_{n-1}$.

Step 2 Select the edge $s \in Q^0_{n-1} \to s'' \in Q^1_{n-1}$ with s'' the unique neighbour of s in Q^1_{n-1}. Since $w(s) = i$ and the δ-th bit of s is 0, s'' satisfies γ_i; we have $w(s'') = i+1$.

Step 3 Select the $k-1$ neighbours $v_1, v_2, \ldots, v_{k-1}$ of s in Q^0_{n-1} that satisfy γ_i. Since $w(s) = i$, we have $w(v_j) = i+1$ ($1 \leqslant j \leqslant k-1$). For an arbitrary bit position z such that the z-th bit of s is set to 1, select the $k-1$ nodes $u_1, u_2, \ldots, u_{k-1}$ in Q^0 with $u_j = v_j \oplus 2^z$ ($1 \leqslant j \leqslant k-1$). Obviously, $w(u_j) = i$ ($1 \leqslant j \leqslant k-1$) holds. Then, select the edges $u_j \to u'_j$ in Q^1_{n-1} ($1 \leqslant j \leqslant k-1$). The nodes $s'', u'_1, u'_2, \ldots, u'_{k-1}$ are adjacent to the node $s' = (s \oplus 2^z) \oplus 2^\delta$. Obviously, we have $w(s') = i$ and $w(u'_j) = i+1$ ($1 \leqslant j \leqslant k-1$).

Step 4 Apply this algorithm recursively in Q^1_{n-1} to find $k-1$ internally disjoint paths $s' \overset{\gamma}{\rightsquigarrow} d$ satisfying $\gamma_{i-1} = (i-1, i)$.

Now, we distinguish two sub-cases depending on the value of $w(d)$.

Case 1.A: $w(d) = i+1$.
The configuration in this case is given in Figure 3.

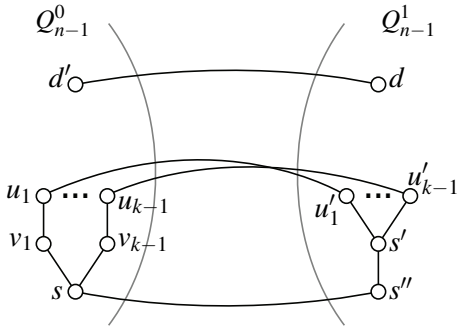

Figure 3: Illustrating the case $w(s) = i$, $w(d) = i+1$.

Step 5 Select the edge $d \to d'$ with d' the unique neighbour of d in Q^0_{n-1}. Find a path $s \overset{\gamma}{\rightsquigarrow} d'$ in Q^0_{n-1} as follows. Select a shortest path $s \overset{\gamma,\,\mathrm{spr}}{\rightsquigarrow} d'$ in Q^0_{n-1}. Let v be the closest node to d' on that path such that v is already included in a path $s \to v_j \to u_j$, say $s \to v_1 \to u_1$. So, s is connected to d' with the path $s \overset{\gamma}{\rightsquigarrow} v \overset{\gamma,\,\mathrm{spr}}{\rightsquigarrow} d'$ with $s \overset{\gamma}{\rightsquigarrow} v$ a sub-path of $s \to v_1 \to u_1$. See Figure 4.

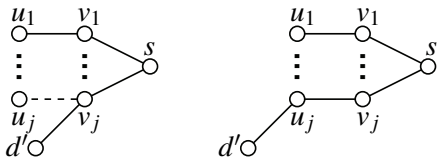

Figure 4: Possible collisions between the path $s \overset{\gamma}{\rightsquigarrow} d'$ and a path $s \overset{\gamma}{\rightsquigarrow} u_j$ in Q^0_{n-1}: part of $s \overset{\gamma}{\rightsquigarrow} u_j$ may be discarded.

Step 6 Assume without loss of generality that d' is connected to s in Q^0_{n-1} via $v_1 \in N(s)$; then the edge $u_1 \to u'_1$ cannot be selected. The paths selected in Q^1_{n-1} are connected to s as follows.

First, assume that the edge $s' \to s''$ is included in one of the selected paths. So, for this particular path, simply replace the edge $s' \to s''$ by $s \to s''$. Assume without loss of generality that there exists a node $u'_w \in Q^1_{n-1}$ neighbour of s' that is not included in any path selected in Q^1_{n-1}. If there is no such node, it means that a path $s' \overset{\gamma}{\rightsquigarrow} d$ selected in Q^1_{n-1} includes two nodes of $N(s')$, say u'_1, u'_2, and thus that path can be shortcut from $s' \overset{\gamma}{\rightsquigarrow} u'_1 \overset{\gamma}{\rightsquigarrow} u'_2 \to d$ to $s' \overset{\gamma}{\rightsquigarrow} u'_1 \to d$, freeing such a node u'_w (here $u'_w = u'_2$).

If the edge $s' \to u'_1$ is included in one of the selected paths, that path $s' \to u'_1 \overset{\gamma}{\rightsquigarrow} d$ is modified to $s \to v_w \to u_w \to u'_w \to s' \to u'_1 \overset{\gamma}{\rightsquigarrow} d$. Each of all the other $k-2$ paths $s' \to u'_j \overset{\gamma}{\rightsquigarrow} d$ is modified to $s \to v_j \to u_j \to u'_j \overset{\gamma}{\rightsquigarrow} d$ ($2 \leqslant j \leqslant k-1$). And otherwise, each path $s' \to u'_j \overset{\gamma}{\rightsquigarrow} d$ is modified to $s \to v_j \to u_j \to u'_j \overset{\gamma}{\rightsquigarrow} d$ ($1 \leqslant j \leqslant k-1$), with the exception in the case $s = d'$ (i.e. $d = s''$) that the path $s' \to d$ of length one is modified to

$s \to v_w \to u_w \to u'_w \to s' \to d$ instead.

Assume the edge $s' \to s''$ is not included in one of the selected paths in Q^1_{n-1}. The path $s' \to u'_1 \overset{\gamma}{\rightsquigarrow} d$ is modified to $s \to s'' \to s' \to u'_1 \overset{\gamma}{\rightsquigarrow} d$, and the other $k-2$ paths $s' \to u'_j \overset{\gamma}{\rightsquigarrow} d$ are modified to $s \to v_j \to u_j \to u'_j \overset{\gamma}{\rightsquigarrow} d$ $(2 \leqslant j \leqslant k-1)$.

Case 1.B: $w(d) = i$.
The configuration in this case is given in Figure 5.

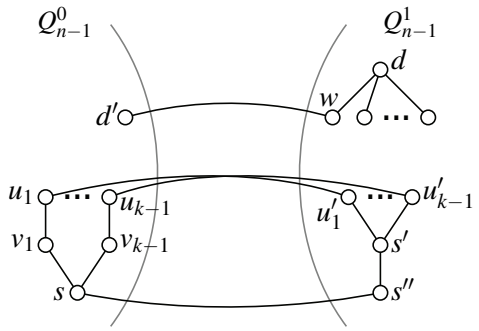

Figure 5: Illustrating the case $w(s) = w(d) = i$.

Step 5 Assume without loss of generality that there exists a node $w \in Q^1_{n-1}$ neighbour of d that is not included in any path selected in Q^1_{n-1}. If there is no such node, it means that a path $s' \overset{\gamma}{\rightsquigarrow} d$ selected in Q^1_{n-1} includes two nodes of $N(d)$, say w_1, w_2, and thus that path can be shortcut from $s' \overset{\gamma}{\rightsquigarrow} w_1 \overset{\gamma}{\rightsquigarrow} w_2 \to d$ to $s' \overset{\gamma}{\rightsquigarrow} w_1 \to d$, freeing such a node w (here $w = w_2$).

Select the path $d \to w \to d'$ with d' the unique neighbour of w in Q^0_{n-1}. Find a path $s \overset{\gamma}{\rightsquigarrow} d'$ in Q^0_{n-1} as in Step 5 of Case A.

Step 6 Similarly, the path $s \overset{\gamma}{\rightsquigarrow} d'$ in Q^0_{n-1} may trigger rerouting in Q^1_{n-1} around s'. This is handled as in Step 6 of Case A.

Algorithm 2 CASE1(Q_n, i, k, s, d)

Input: A Q_n, a bit constraint $\gamma_i = (i, i+1)$, k the number of paths to find ($k \leqslant \min(n-i, i+1)$), a source node s with $w(s) = i$ and a destination node d.
Output: $k = \min(n-i, i+1)$ internally node-disjoint paths $s \overset{\gamma}{\rightsquigarrow} d$ in Q_n satisfying γ_i.

$R := \emptyset$; *// Result set*
$\delta := \lfloor \log((s \oplus d) \,\&\, d) \rfloor$;
$z := \lfloor \log s \rfloor$;
$s'' := s \oplus 2^\delta$;
$s' := s \oplus z \oplus 2^\delta$;
// The simple cases $s' = d$ and $s'' = d$ are omitted.
$\{v_1, v_2, \ldots, v_{k-1}\} := N(s) \cap Q^0_{n-1}$ sat. γ_i;
$\{u_1, u_2, \ldots, u_{k-1}\} := \{v_1 \oplus z, v_2 \oplus z, \ldots, v_{k-1} \oplus z\}$;
$\{u'_1, u'_2, \ldots, u'_{k-1}\} := \{u_1 \oplus 2^\delta, u_2 \oplus 2^\delta, \ldots, u_{k-1} \oplus 2^\delta\}$;
$P :=$ HC-CONTAINER(Q^1_{n-1}, $i-1$, $k-1$, s', d);
if $w(d) = i+1$ **then**
  $d' := d \oplus 2^\delta$;
  $(p^* : s' \to u'_x \overset{\gamma}{\rightsquigarrow} d') :=$ HC-SPR(Q^0_{n-1}, i, s, d');
  for all $p : s' \to u'_j \overset{\gamma}{\rightsquigarrow} d$ in P **do**
    if $u'_j = u'_x$ **then**
      if $\exists q \in P$ with $(s' \to s'') \in q$ **then**
        $u'_w := \{u'_1, u'_2, \ldots, u'_j\} \setminus \bigcup_{q \in P} q$;
        $R := R \cup \{s \to v_w \to u_w \to u'_w \to p\}$
      else
        $R := R \cup \{s \to s'' \to p\}$
    else
      if $u'_j = s''$ **then**
        $R := R \cup \{s \to (u'_j \overset{\gamma}{\rightsquigarrow} d) \subset p\}$
      else
        $R := R \cup \{s \to v_j \to u_j \to (u'_j \overset{\gamma}{\rightsquigarrow} d) \subset p\}$
else // $w(d) = i$
  $w := (N(d) \cap Q^1_{n-1}$ sat. $\gamma_i) \setminus \bigcup_{q \in P} q$;
  $d' := w \oplus 2^\delta$;
  $(p^* : s' \to u'_x \overset{\gamma}{\rightsquigarrow} d') :=$ HC-SPR(Q^0_{n-1}, i, s, d');
  for all $p : s' \to u'_j \overset{\gamma}{\rightsquigarrow} d$ in P **do**
    if $u'_j = u'_x$ **then**
      if $\exists q \in P$ with $(s' \to s'') \in q$ **then**
        $u'_w := \{u'_1, u'_2, \ldots, u'_j\} \setminus \bigcup_{q \in P} q$;
        $R := R \cup \{s \to v_w \to u_w \to u'_w \to p\}$
      else
        $R := R \cup \{s \to s'' \to p\}$
    else
      if $u'_j = s''$ **then**
        $R := R \cup \{s \to (u'_j \overset{\gamma}{\rightsquigarrow} d) \subset p\}$
      else
        $R := R \cup \{s \to v_j \to u_j \to (u'_j \overset{\gamma}{\rightsquigarrow} d) \subset p\}$
return $R \cup \{p^* \to d\}$

3.1.2. Case 2: $w(s) = i+1$

We first proceed in two steps before reducing this case to Case 1 (i.e. Section 3.1.1). Pseudo-code is given in Algorithm 3.

Step 1 Find a bit position δ ($0 \leqslant \delta \leqslant n-1$) such that the δ-th bit of s is set to 1 and the δ-th bit of d is set to 0.

Reducing the hypercube Q_n along this bit position δ, we obtain the two subcubes Q^0_{n-1} and Q^1_{n-1}, and $s \in Q^1_{n-1}$, $d \in Q^0_{n-1}$.

Step 2 Select the edge $s \in Q^1_{n-1} \to s'' \in Q^0_{n-1}$ with s'' the unique neighbour of s in Q^0_{n-1}. Since $w(s) = i+1$ and the δ-th bit of s is set to 1, s'' satisfies γ_i; we have $w(s'') = i$.

Then, the case $w(s) = i+1$, $w(d) = i$ is solved similarly to the case $w(s) = i$, $w(d) = i+1$ (Case 1.A) by exchanging the roles of s and d, and the case $w(s) = i+1$, $w(d) = i+1$ is solved similarly to the case $w(s) = i$, $w(d) = i$ (Case 1.B) with the differences that

- in Step 3, the bit position z is selected such that the z-th bit of s is set to 0;
- in Step 4, the algorithm is applied recursively in Q^0_{n-1}, the constraint considered thus remaining $\gamma_i = (i, i+1)$;
- in Step 5, a shortest-path routing algorithm is applied in Q^1_{n-1}, the constraint considered thus becoming $\gamma_{i-1} = (i-1, i)$.

Algorithm 3 CASE2(Q_n, i, k, s, d)

Input: A Q_n, a bit constraint $\gamma_i = (i, i+1)$, k the number of paths to find ($k \leqslant \min(n-i, i+1)$), a source node s with $w(s) = i+1$ and a destination node d.

Output: $k = \min(n-i, i+1)$ internally node-disjoint paths $s \overset{\gamma}{\rightsquigarrow} d$ in Q_n satisfying γ_i.

1: **if** $w(d) = i$ **then**
2: **return** HC-CONTAINER(Q_n, i, k, d, s)
3: $R := \emptyset$; *// Result set*
4: $\delta := \lfloor \log((s \oplus d) \,\&\, s) \rfloor$;
5: $z := \lfloor \log \neg s \rfloor$;
6: $s'' := s \oplus 2^\delta$;
7: $s' := s \oplus z \oplus 2^\delta$;
8: $\{v_1, v_2, \ldots, v_{k-1}\} := N(s) \cap Q^1_{n-1}$ sat. γ_i;
9: $\{u_1, u_2, \ldots, u_{k-1}\} := \{v_1 \oplus z, v_2 \oplus z, \ldots, v_{k-1} \oplus z\}$;
10: $\{u'_1, u'_2, \ldots, u'_{k-1}\} := \{u_1 \oplus 2^\delta, u_2 \oplus 2^\delta, \ldots, u_{k-1} \oplus 2^\delta\}$;
11: $P :=$ HC-CONTAINER(Q^0_{n-1}, i, $k-1$, s', d);
12: $w := (N(d) \cap Q^1_{n-1}$ sat. $\gamma_i) \setminus \bigcup_{q \in P} q$;
13: *// The simple cases $s' = d$ and $s'' = w$ are omitted.*
14: $d' := w \oplus 2^\delta$;
15: $(p^* : s' \to u'_x \overset{\gamma}{\rightsquigarrow} d') :=$ HC-SPR(Q^1_{n-1}, $i-1$, s, d');
16: **for all** $p : s' \to u'_j \overset{\gamma}{\rightsquigarrow} d$ in P **do**
17: **if** $u'_j = u'_x$ **then**
18: **if** $\exists q \in P$ with $(s' \to s'') \in q$ **then**
19: $u'_w := \{u'_1, u'_2, \ldots, u'_j\} \setminus \bigcup_{q \in P} q$;
20: $R := R \cup \{s \to v_w \to u_w \to u'_w \to p\}$
21: **else**
22: $R := R \cup \{s \to s'' \to p\}$
23: **else**
24: **if** $u'_j = s''$ **then**
25: $R := R \cup \{s \to (u'_j \overset{\gamma}{\rightsquigarrow} d) \subset p\}$
26: **else**
27: $R := R \cup \{s \to v_j \to u_j \to (u'_j \overset{\gamma}{\rightsquigarrow} d) \subset p\}$
28: **return** $R \cup \{p^* \to w \to d\}$

3.2. Correctness and complexities

We show in this section the correctness of the algorithm of Section 3.1 and establish its worst case time and path length complexities.

Lemma 2. *The algorithm of Section 3.1 is correct and always terminates.*

Proof. We recall that the edge $s \to s''$ is selected, with s, s'' in distinct subcubes. We start by showing the existence of a reduction bit δ.

Assume $w(s) = i$. Suppose there is no bit position δ with the δ-th bit of s set to 0 and the δ-th bit

of d set to 1. If $w(d) = i$, this supposition implies that $s = d$, which is a contradiction. If $w(d) = i+1$, this supposition implies that all the bits of d corresponding to the positions of the $n-i$ bits of s set to 0 are also set to 0, and thus that d has $n-i$ bits set to 0 and $i+1$ bits set to 1. This is a contradiction since d of n bits ($d \in Q_n$).

Assume $w(s) = i+1$. Suppose there is no bit position δ with the δ-th bit of s set to 1 and the δ-th bit of d set to 0. If $w(d) = i+1$, this supposition implies that $s = d$, which is a contradiction. If $w(d) = i$, this supposition implies that all the bits of s corresponding to the positions of the $n-i$ bits of d set to 0 are also set to 0, and thus that s has $n-i$ bits set to 0 and $i+1$ bits set to 1. This is a contradiction since s of n bits ($s \in Q_n$).

Regarding the existence of available neighbours and rerouting feasibility, a proof has already been given in the corresponding steps of Section 3 for more clarity. □

Lemma 3. *The algorithm of Section 3.1 generates internally node-disjoint paths of lengths at most $n+3k$ in $O(kn)$ time.*

Proof. The paths generated by the algorithm of Section 3.1 are internally node-disjoint as shown by the algorithm description.

We now consider the maximum length of a generated path. A total of $k = \min(n-i, i+1)$ paths are generated. First, one should note that exactly one hypercube reduction is required for each path to be generated. So, in total, the original hypercube Q_n is reduced k times. In other words, routing is performed inside hypercubes of successive dimensions $n, n-1, \ldots, n-k$. When the base case condition $k = 1$ is satisfied a shortest-path routing algorithm is applied. This base case $k = 1$ actually means that either $i = 0$ and thus a path of length at most two is generated, or $i = n-1$ and thus a path of length at most $n-k$ is generated. In addition, each hypercube reduction induces $3-1 = 2$ additional edges to connect the paths selected by induction inside one subcube to the node s located inside the other subcube. Rerouting triggered in Step 6 may induce an extra two edges to connect a path, say $u'_1 \overset{\gamma}{\rightsquigarrow} d$, in the sub-cube of d to s via the special node $u'_w \in N(s')$ (precisely, the two extra edges are $u'_w \to s' \to u'_1$). Thus, at most four edges in total to be added at each reduction to connect paths in the sub-cube of d to s. Therefore, generated paths have lengths of at most $4k + (n-k) = n+3k$ edges. The single path connecting d to s by application of a shortest-path routing algorithm inside the subcube of s requires at most two edges for the sub-path $d \overset{\gamma}{\rightsquigarrow} d'$ and at most $n-1$ edges for the shortest path $d' \overset{\gamma,\,\mathrm{spr}}{\rightsquigarrow} s$, thus requiring in total at most $n+1$ edges.

Regarding the time complexity of the algorithm of Section 3.1, Steps 1 and 2 are both constant time $O(1)$. Step 3 is linear time $O(n)$. Let $T(n)$ be the time required to solve the problem in a Q_n. Step 4 is thus $T(n-1)$ time. Steps 5 and 6 are both linear time $O(n)$. From this discussion, we obtain the equation $T(n) = T(n-1) + O(n)$. Since we have exactly k hypercube reductions, the total time complexity of the proposed algorithm is $O(kn)$. □

So, we can summarise this discussion in the following theorem.

Theorem 4. *In a Q_n, given a bit constraint $\gamma_i = (i, i+1)$ and any two distinct nodes s and d satisfying γ_i, we can select $k = \min(n-i, i+1)$ internally node-disjoint paths $s \overset{\gamma}{\rightsquigarrow} d$ satisfying γ_i and of lengths at most $n+3k$ in $O(kn)$ time.*

Proof. This can be directly deduced from Lemmas 2 and 3. □

3.3. Routing example

In a Q_5, given a bit constraint $\gamma_2 = (2,3)$, a source node s : 11010 and a destination node d : 10101, an execution trace of the algorithm of Section 3 is given in Table 1. As a result, the following three internally node-disjoint paths, all satisfying γ_2, are selected:

- $s = 11010 \to 11000 \to 11100 \to 10100 \to 10101 = d$
- $s = 11010 \to 10010 \to 10011 \to 10001 \to 10101 = d$
- $s = 11010 \to 01010 \to 01110 \to 00110 \to 00111 \to 00101 \to 10101 = d$

Table 1: Node-to-node disjoint paths routing example in a Q_5 with bit constraint $\gamma_2 = (2,3)$.

n	2^δ	s	d	s'	d'	$\in Q^0_{n-1}$	$\in Q^1_{n-1}$
5	8	11010	10101	10110	-	d	s
Selection of sub-path $s = 11010 \rightarrow 1010 \rightarrow 1110$. *Induction on* Q^0_4.							
4	2	10110	10101	10101	-	d, s'	s
Selection of sub-path $s = 10110 \rightarrow 10010 \rightarrow 10011$. *Induction on* Q^0_3*:* $s' = d$*, and thus selection of* $s = 10110 \rightarrow 10010 \rightarrow 10011 \rightarrow 10001 \rightarrow 10101 = d$.							
4	2	10110	10101	10101	00111	d, s'	s, d'
Selection of $s = 10110 \rightarrow 00110 \rightarrow 00111 \rightarrow 00101 \rightarrow 10101 = d$ *(by shortest-path routing in* Q^1_3*).*							
5	8	11010	10101	10110	11100	d, s'	s, d'
Selection of $s = 11010 \rightarrow 11000 \rightarrow 11100 \rightarrow 10100 \rightarrow 10101 = d$ *(by shortest-path routing in* Q^1_4*),* $s = 11010 \rightarrow 01010 \rightarrow 01110 \rightarrow 00110 \rightarrow 00111 \rightarrow 00101 \rightarrow 10101 = d$*, and* $s = 11010 \rightarrow 10010 \rightarrow 10011 \rightarrow 10001 \rightarrow 10101 = d$ *(by joining sub-paths).*							

4. Empirical evaluation

In order to inspect the practical behaviour of the proposed hypercube node-to-node disjoint paths routing algorithm, that is in other words investigating how the algorithm performs in average, we have conducted an empirical evaluations consisting of various measurements. Additionally, we aimed at comparing the obtained experimental results to the theoretical estimations of Section 3.2.

For this experimentation, we have implemented the algorithm HC-CONTAINER of Section 3 using the Scheme functional programming language[21]. We have then considered a hypercube of dimension n with $4 \leqslant n \leqslant 16$ and a bit constraint $\gamma_2 = (2,3)$, and we have used our algorithm implementation to solve 10,000 random instances of the container problem for each value of n. One should note that values of n smaller than 4 are ignored since at most $k = \min(n-i, i+1)$ disjoint paths can be found, that is at most one path for $n < 4$, and thus a shortest-path routing algorithm would suffice in such case. The source node and destination node were randomly selected from the set of the nodes satisfying the bit constraint γ_2 in Q_n; they are distinct.

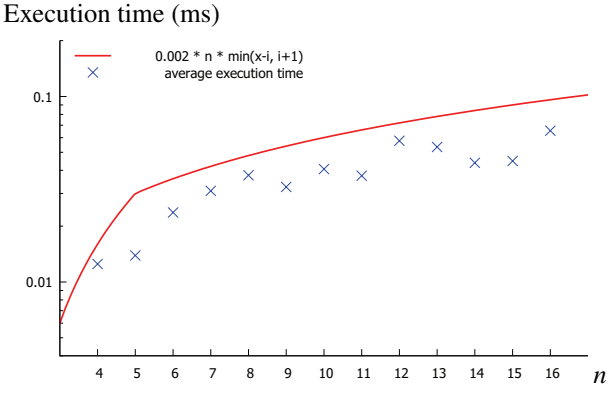

Figure 6: Average execution time for each value of n ($i = 2$ and $4 \leqslant n \leqslant 16$).

So first, we have measured the average time required to solve one instance of the container problem in the conditions previously described. The results are given in Figure 6. The theoretical worst-case time complexity established in Section 3.2 has also been plotted for reference.

Then, we have measured the maximum path length obtained for each value of n, and also for each value of n the average of the 10,000 maximum path lengths, each average value being obtained when solving one instance of the container problem. The results are illustrated in Figure 7. To facilitate comparison with the theoretical estimation, we have additionally plotted the theoretical maximum path length as established in Lemma 3.

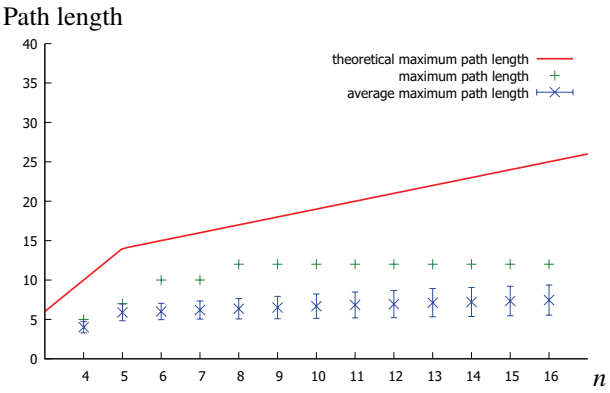

Figure 7: Maximum path length and average maximum path length with standard deviation for each value of n ($i = 2$ and $4 \leqslant n \leqslant 16$).

First, one can observe that the time complexity theoretically established is not overestimated as it follows the average algorithm execution time. Second, one can notice that our algorithms performs significantly better than the theoretical estimation for the maximum path length, paths being in practice much shorter. There may thus be room for refining our theoretical upper bound on the length of a generated path. Also, one can see that the maximum maximal path length recorded seems to stabilise beyond $n = 8$. This is can be explained by the fact that since both the source node and the destination node satisfy the constraint γ_i, the Hamming distance between them is bounded by $i + (i+1)$, that is 5 in our experiment.

5. Conclusions

Thanks to simplicity for both hardware and software implementation, hypercubes are popular interconnection networks for massively parallel systems. Disjoint paths routing is one robust method to avoid infamous parallel systems resource allocations problems such as deadlocks or starvations. In this paper, we have presented an algorithm solving the container problem with bit constraint in a hypercube Q_n. For a bit constraint $\gamma_i = (i, i+1)$ and any two distinct nodes s, d satisfying γ_i, the algorithm selects $k = \min(n-i, i+1)$ internally node-disjoint paths $s \overset{\gamma}{\rightsquigarrow} d$ satisfying γ_i. The lengths of the selected paths are at most $n+3k$, and the time complexity of this routing algorithm is $O(kn)$. We have analysed empirically the average behaviour of the proposed algorithm, the obtained results showing that regarding the lengths of the generated paths, this algorithm performs in practice significantly better than the theoretical worst-case estimations. Consequently, by enforcing a bit constraint when routing, this algorithm enables the selection of several sets of disjoint paths between several node pairs, each pair satisfying a distinct bit constraint. Conventional routing algorithms, even disjoint paths routing algorithms, are not able to provide such result.

Regarding future works, extending this research to solve the node-to-set disjoint paths routing problem with bit constraint in a hypercube is a meaningful objective. In addition, enhancing the fault tolerance for instance by considering cluster-fault tolerance, is also an interesting objective.

Acknowledgments

This study was partly supported by a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science under Grant No. 25330079.

1. Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Transactions on Computers*, **37**(7), 867–872 (1988).
2. C. L. Seitz, "The cosmic cube," *Communications of the ACM*, **28**(1):22–33 (1985).
3. TOP500, "TOP500 List November 2014," `http://top500.org/list/2014/11/`, November 2014. Last accessed April 2015.
4. Y. Li, S. Peng and W. Chu, "Efficient collective communications in dual-cube," *The Journal of Supercomputing*, **28**(1):71–90 (2004).
5. Y. Li, S. Peng and W. Chu, "Metacube - a versatile family of interconnection networks for extremely large-scale supercomputers," *The Journal of Supercomputing*, **53**(2):329–351 (2010).
6. Q. M. Malluhi and M. A. Bayoumi, "The hierarchical hypercube: a new interconnection topology for massively parallel systems," *IEEE Transactions on Parallel and Distributed Systems*, **5**(1):17–30 (1994).
7. K. Ghose and K. R. Desai, "The HCN: a versatile interconnection network based on cubes," *In Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, pp. 426–435, Reno, NV, USA, November 12–17 (1989).
8. S. Gao, B. Novick and K. Qiu, "From Hall's Matching Theorem to Optimal Routing on Hypercubes," *Journal of Combinatorial Theory, Series B*, **74**:291–301 (1998).
9. O. Sinanoglu, M. H. Karaata and B. AlBdaiwi, "An inherently stabilizing algorithm for node-to-node routing over all shortest node-disjoint paths in hypercube networks," *IEEE Transactions on Computers*, **59**(7):995–999 (2010).
10. A. Bossard and K. Kaneko, "Time optimal node-to-set disjoint paths routing in hypercubes," *Journal of Information Science and Engineering*, **30**(4):1087–1093 (2014).
11. Q.-P. Gu, S. Okawa and S. Peng, "Set-to-set fault tolerant routing in hypercubes," *IEICE Transactions on Fundamentals*, **E79-A**(4):483–488 (1996).
12. Q.-P. Gu and S. Peng, "An efficient algorithm for the k-pairwise disjoint paths problem in hypercubes," *Journal of Parallel and Distributed Computing*, **60**(6):764–774 (2000).
13. A. Bossard and K. Kaneko, "On hypercube routing and fault tolerance with bit constraint," *In Proceedings of the Second International Symposium on Computing and Networking*, pp. 40–49, Shizuoka City, Japan, December 10–12 (2014).
14. Y. Li, S. Peng and W. Chu, "Disjoint paths in metacube," *In Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 43–50, Marina del Rey, CA, USA, November 3–5 (2003).
15. S. Murugesan, "Harnessing Green IT: Principles and practices," *IT Professional*, **10**(1):24–33 (2008).
16. J. Chen, I. A. Kanj and G. Wang, "Hypercube network fault tolerance: a probabilistic approach," *Journal of Interconnection Networks*, **6**(1):17–34 (2005).
17. M. Dietzfelbinger, S. Madhavapeddy and I. H. Sudborough, "Three disjoint path paradigms in star networks," *In Proceedings of the Third IEEE Symposium*

on Parallel and Distributed Processing, pp. 400–406, Dallas, TX, USA, December 2–5 (1991).
18. Y. Suzuki and K. Kaneko, "An algorithm for node-disjoint paths in pancake graphs," *IEICE Transactions on Information and Systems*, **E86-D**(3):610–615 (2003).
19. K. Kaneko and N. Sawada, "An algorithm for node-to-node disjoint paths problem in burnt pancake graphs," *IEICE Transactions on Information and Systems*, **E90-D**(1):306–313 (2007).
20. K. Menger, "Zur allgemeinen Kurventheorie," *Fundamenta Mathematicae*, **10**:96–115 (1927).
21. R. B. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler and M. Felleisen, "DrScheme: a programming environment for scheme," *Journal of Functional Programming*, **12**(2):159–182 (2002).