ELSEVIER Theoretical Computer Science 270 (2002) 111–124

Theoretical Computer Science

www.elsevier.com/locate/tcs

Some permutation routing algorithms for low-dimensional hypercubes

F.K. Hwang[a], Y.C. Yao[b,*], B. Dasgupta[c]

[a]Department of Applied Mathematics, National Chiao-Tung University, Hsin-Chu 30050, Taiwan, Republic of China

[b]Institute of Statistical Science, Academia Sinica, Taipei, Taiwan, Republic of China

[c]Department of Computer Science, University of Illinois, Chicago, IL 60607, USA

Received 25 January 2000; accepted 22 June 2000

Communicated by D.-Z. Du

Abstract

Oblivious permutation routing in binary d-cubes has been well studied in the literature. In a permutation routing, each node initially contains a packet with a destination such that all the 2^d destinations are distinct. Kaklamanis et al. (Math. Syst. Theory 24 (1991) 223–232) used the decomposability of hypercubes into Hamiltonian circuits to give an asymptotically optimal routing algorithm. The notion of "destination graph" was first introduced by Borodin and Hopcroft to derive lower bounds on routing algorithms. This idea was recently used by Grammatikakis et al. (Proceedings of the Advancement in Parallel Computing, Elsevier, Amsterdam, 1993) to construct many–one routing algorithms for the binary 2-cube and 3-cube. In the present paper, further theoretical development is made along this line. It is then applied to obtain algorithms for binary d-cubes with d up to 12, which compare favorably with the above-mentioned "Hamiltonian circuit" algorithm. Some results on t-nary cubes with $t \geqslant 3$ are also obtained. © 2002 Published by Elsevier Science B.V.

Keywords: Permutation routing; Hypercube; Oblivious routing; Destination graph; t-nary cube

1. Introduction

Let $V = \{(x_1,\dots,x_d) : x_i \in \{0,1,\dots,t-1\}, i = 1,\dots,d\}$ denote the set of t^d nodes of a t-nary d-cube such that for any two nodes $u, v \in V$, there is a link from u to v if their Hamming distance equals 1 (hence also a link from v to u). Thus, at each node, there are $d(t-1)$ in-links and $d(t-1)$ out-links. In a permutation routing, each node u initially contains a packet with destination $D(u) \in V$ such that $\{D(u) : u \in V\} = V$.

* Corresponding author.
E-mail address: yao@stat.sinica.edu.tw (Y.C. Yao).

Under the multiport model, at each step a packet can either stay put or move to an adjacent node by crossing a link, but no link can be crossed by two packets at the same step. Cohabitation of multiple packets at the same node is allowed. The goal is to minimize the number of steps required to route all packets to their respective destinations. Since the diameter of the t-nary d-cube equals d, d is a (worst-case) lower bound on the required number of steps. A routing algorithm is called *tight* if it requires at most d steps. An algorithm is called *minimum routing* if a packet can move from a node to an adjacent one only when the move corrects an incorrect digit (implying that the total number of steps required for a packet with origin u equals the Hamming distance between u and $D(u)$ plus the number of staying-put steps).

Besides minimizing the number of steps, a routing algorithm must also be easy to implement; namely, the routing at each step should be determined efficiently. One such class, called *oblivious* routing (see [8]), has been extensively studied in the literature. For an oblivious algorithm, the routing of a packet is determined only by its origin and destination. In other words, an oblivious algorithm specifies, for each pair of nodes u and v, a path $\mathscr{P}_{uv}$ from u to v, such that a packet at node u with destination $D(u)$ moves along the path $\mathscr{P}_{uD(u)}$. Let "+" denote addition modulo t. An oblivious algorithm $\{\mathscr{P}_{uv} : u, v \in V\}$ is called *translation invariant* (or simply *invariant*) if $\mathscr{P}_{(u+w)(v+w)} = \mathscr{P}_{uv} + w$ for all $u, v, w \in V$ where the "+" in $\mathscr{P}_{uv} + w$ means that the "$+w$" operation is applied to each node in the path $\mathscr{P}_{uv}$. Thus, an invariant oblivious algorithm is completely determined by paths $\mathscr{P}_{uv}$ with all $u \in V$ and $v = \mathbf{0} \equiv (0, \ldots, 0)$.

Since packets cohabiting at a node require a buffer to store, it is desirable to keep their number small. However, this issue is not addressed in the present paper as the worst-case analysis of the required buffer size is a very difficult problem.

Borodin and Hopcroft [1] introduced the notion of *destination graph* (DG) for oblivious algorithms. A DG associated with a given node v is the union of paths from all origins to the destination v (i.e. $\bigcup_{u \in V} \mathscr{P}_{uv}$). In general, an oblivious algorithm requires to specify t^d DGs, one for each $v \in V$. However, an invariant oblivious algorithm is determined by only one DG with $v = \mathbf{0}$, which will be referred to as the *modular* DG.

Borodin and Hopcroft used DG to derive a lower bound $\Omega(\sqrt{n}/\Delta^{3/2})$ for oblivious algorithms on the number of steps for an n-node Δ-indegree graph. Applying to the t-nary d-cube yields a bound $\Omega(t^{(d-3)/2}/d^{3/2})$. For the binary d-cube, they gave an $O(2^{d/2})$ oblivious algorithm by dividing the routing into two subroutings, one on the first $d/2$ dimensions and the other on the last $d/2$ dimensions (the former subrouting is no longer permutation but many–one whereas the latter is one–many). Kaklamanis et al. [6] improved the $\Omega(\sqrt{n}/\Delta^{3/2})$ lower bound to $\Omega(\sqrt{n}/\Delta)$, which was later shown to be sharp by Borodin et al. [2]. For the binary d-cube, Kaklamanis et al. [6] gave an $O(2^{d/2}/d)$ oblivious algorithm by using the decomposability of binary hypercubes into Hamiltonian circuits. They commented that their algorithm may be practical for small d.

Indeed, in practical applications, hypercubes are mostly binary with relatively small d. Thus, it is of special interest to find the best routing algorithm for such networks. Note that the algorithm of Kaklamanis, Krizanc and Tsantilas is neither minimum routing nor tight, hence subject to improvement for small d. In this respect, Hwang et al. [5] gave tight minimum-routing algorithms for binary d-cubes with $d \leqslant 7$, and Grammatikakis et al. [3] gave tight oblivious minimum-routing algorithms for binary d-cubes with $d \leqslant 6$. In particular, the latter showed a novel use of DG for constructing a tight oblivious algorithm for the 6-cube.

In Section 2, we develop a theory of constructing invariant oblivious algorithms based on DG. This enables us to obtain in Section 3 tight invariant oblivious minimum-routing algorithms for the binary 7- and 8-cube, and a 11(14,19,24)-step invariant oblivious minimum-routing algorithm for the 9(10,11,12)-cube. We also consider routing algorithms for the t-nary d-cube with $d \leqslant 4$. In the special case $t = 3, d = 4$, Hwang et al. [5] obtained a complicated tight invariant oblivious minimum-routing algorithm, whereas we provide a very simple one using DG with little effort. It is also shown that there exists no tight invariant oblivious algorithm for permutation routing in the t-nary 3-cube with $t \geqslant 4$.

2. The DG method from a tree viewpoint

In this section, we give results on using DG to construct invariant oblivious algorithms for permutation routing as well as many–one routing where the latter means that all packets have distinct origins but not necessarily distinct destinations.

For both permutation and many–one routings, it is clear that no competition for the same link from two packets can occur at step 1. Moreover, for permutation routing, if after a number of steps, all packets have at most one incorrect digit, then one more step enables all packets to reach their destinations.

Since we shall only be concerned with invariant oblivious algorithms, it suffices to consider a modular DG. A modular DG is viewed as a rooted tree with the root labeled by the node $\mathbf{0}$, and $t^d - 1$ leaves each labeled by a distinct node of V other than $\mathbf{0}$, such that the path from a leaf labeled u to the root is $\mathscr{P}_{u\mathbf{0}}$. (Note that we shall make a conscious distinction between a node and a vertex by referring to a point of V as a node and a point of a DG as a vertex.) When two paths agree on their last, say, k steps, we shall merge this portion of the paths so as to simplify the expression of the modular DG. Note that a simplified DG may have fewer than $t^d - 1$ leaves. As an example, the two modular DGs in Figs. 1 (a) and (b) are equivalent. In Fig. 1(b), an edge connecting two vertices labeled by the same node $(0,0,1)$ indicates staying put at this node. In a simplified DG, for each node $u \in V$, the path from a vertex labeled u to the root specifies $\mathscr{P}_{u\mathbf{0}}$. In case that two (or more) vertices are labeled by u (as in Fig. 1(b) with $u = (0,0,1)$), we would have two candidate paths for $\mathscr{P}_{u\mathbf{0}}$. To avoid such confusion, an "$*$" is attached to indicate the right vertex. Using the invariant algorithm induced by the two equivalent modular DGs in Fig. 1, a packet with origin $(1,0,1)$ and destination $\mathbf{0}$ moves along the path $(1,0,1) \to (0,0,1) \to (0,0,1) \to \mathbf{0}$ in three steps (staying put at step 2), while a packet with origin $(1,1,1)$ and destination $(1,0,0)$ moves (by invariance) along the path $(1,1,1) \to (1,0,1) \to (1,0,0)$ in two steps. Note that in the

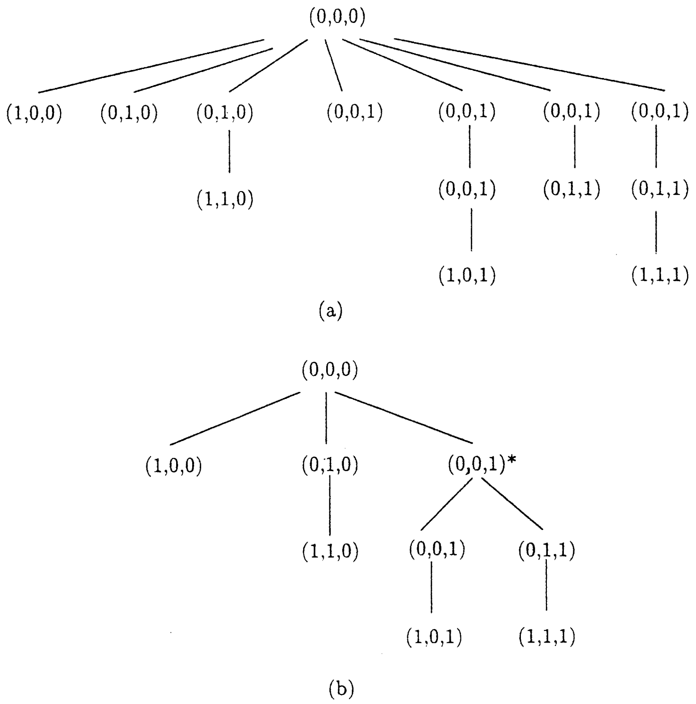

Fig. 1. Two equivalent modular DGs for the binary 3-cube.

many–one routing setting, if two packets have respective origins $(1,0,1)$ and $(1,1,1)$ but common destination $\mathbf{0}$, then they would compete for the link $(0,0,1)\rightarrow\mathbf{0}$ at step 3, so that the modular DG does not induce a valid many–one routing algorithm. However, for a permutation routing, the above situation cannot happen. Indeed, it follows from Lemma 3 below that no competition for the same link from two packets can occur, so that all packets reach their destinations in three steps. A modular DG is said to induce a *valid* permutation (many–one) routing algorithm if competition for the same link from two packets can never occur under the permutation (many–one) routing setting. As another example, the modular DG in Fig. 2 induces, by Lemma 2 below, a valid many–one routing algorithm which requires four steps to route all packets. It should be remarked that if a modular DG induces a valid many–one routing algorithm, then it can be used for the one–many routing setting by reversing the routing steps. For example, based on the DG in Fig. 2, in a many–one routing, a packet with origin $(1,1,0)$ and destination $\mathbf{0}$ moves along the path $(1,1,0)\rightarrow(0,1,0)\rightarrow\mathbf{0}$ and then stays put at steps 3 and 4, while in a one–many routing, a packet with origin $\mathbf{0}$ and destination $(1,1,0)$ moves along the path $\mathbf{0}\rightarrow\mathbf{0}\rightarrow\mathbf{0}\rightarrow(0,1,0)\rightarrow(1,1,0)$. (Note that in the latter case, if the packet moves along the path $\mathbf{0}\rightarrow(0,1,0)\rightarrow(1,1,0)\rightarrow(1,1,0)\rightarrow(1,1,0)$, then the routing is not considered the reverse of the original many–one routing algorithm, and the resulting algorithm may not be valid in the one–many routing setting.)

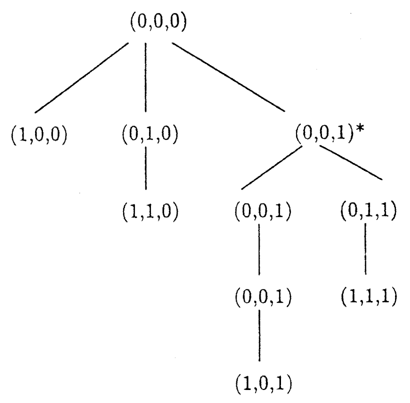

Fig. 2. A modular DG for the binary 3-cube.

Lemma 1. *Every algorithm for many–one routing in the t-nary d-cube requires at least* $\lceil (t^d - 1)/d(t-1) \rceil$ *steps (in the worst case).*

Proof. Consider the case that all packets have the same destination, say $\mathbf{0}$. As $\mathbf{0}$ has $d(t-1)$ in-links, at most $d(t-1)$ packets can reach $\mathbf{0}$ at each step. It follows that $\lceil (t^d - 1)/d(t-1) \rceil$ is a (worst-case) lower bound on the number of steps needed. □

Let T be a modular DG and $p_u \equiv \mathscr{P}_{u\mathbf{0}}$ the path corresponding to node u. Let $|p_u|$ denote the length of p_u. Then $l(T) \equiv \max_{u \in V} |p_u|$ is simply the depth of T, which is the (maximum) number of steps required by the invariant algorithm induced by T provided that no competition for a link from two packets can arise at any step. In the following, we provide in the many–one routing setting a necessary and sufficient condition for modular DGs which guarantees that no two packets compete for the same link at any step.

The root of a modular DG is called a level-0 vertex, its children level-1 vertices, and so on. A level-k edge is an edge from a level-k vertex to a level-$(k-1)$ vertex. For a vertex x, let $u_x^* \in V$ denote the label of x. edge from vertex x to vertex y is said to be of type 0 if $u_x^* = u_y^*$, and of type $(i,j), 1 \leqslant i \leqslant d, 1 \leqslant j \leqslant t-1$ if $u_x^* \neq u_y^*$ and

$$(\text{the } i\text{th digit of } u_x^*) - j = (\text{the } i\text{th digit of } u_y^*) (\text{mod } t)$$

where digits are numbered from left to right. When $t = 2$, type (i,j) will simply be called type i since j always equals 1. Let $S_n(u) \equiv (S_n^{(1)}(u), S_n^{(2)}(u))$, $1 \leqslant S_n^{(1)}(u) \leqslant d$, $1 \leqslant S_n^{(2)}(u) \leqslant t-1$, denote the type of the nth edge of path p_u $(n = 1, \ldots, |p_u|)$ if it is not of type 0; and let $S_n(u) \equiv (0,0)$ if the nth edge is of type 0. We call $S(u) \equiv (S_1(u), \ldots, S_{|p_u|}(u))$ the *type sequence* for path p_u. Note that

$$\sum_{\{m: S_m^{(1)}(u) = i\}} S_m^{(2)}(u) = (\text{the } i\text{th digit of } u) (\text{mod}\, t), \quad i = 1, \ldots, d.$$

Lemma 2. *A modular DG induces a valid many-one routing algorithm if and only if the following Condition* (C1) *holds.*

Condition (C1): *For any two type sequences $S(u)$ and $S(v)$, we have*

$$\sum_{\{m<n: S_m^{(1)}(u)=i\}} S_m^{(2)}(u) = \sum_{\{m<n: S_m^{(1)}(v)=i\}} S_m^{(2)}(v) \,(\text{mod}\, t) \quad \textit{for all } i = 1,\dots,d,$$

whenever $S_n(u)=S_n(v) \neq (0,0)$ for some n.

Proof. (*Sufficiency*) Consider two packets with respective origins $u, v (u \neq v)$, and destinations $D(u), D(v)$. By invariance, the two packets move along the paths $p_{u-D(u)} + D(u)$ and $p_{v-D(v)} + D(v)$, with corresponding type sequences $S(u')$ and $S(v')$, where $u' = u - D(u)$ and $v' = v - D(v)$. Suppose the packets compete for the same link at step n. Thus, $S_n(u') = S_n(v') \neq (0,0)$, and the packets are at the same node, say w, at the beginning of step n. Then we have for $i = 1,\dots,d$,

$$(\text{the } i\text{th digit of } u - w) = \sum_{\{m<n: S_m^{(1)}(u')=i\}} S_m^{(2)}(u') \,(\text{mod}\, t),$$

$$(\text{the } i\text{th digit of } v - w) = \sum_{\{m<n: S_m^{(1)}(v')=i\}} S_m^{(2)}(v') \,(\text{mod}\, t).$$

By Condition (C1), $u = v$, a contradiction.

(*Necessity*) Let T be a modular DG which does not satisfy Condition (C1), i.e. there exist two type sequences $S(u)$ and $S(v)$ such that $S_n(u) = S_n(v) \neq (0,0)$, but $\sigma_i(u) \neq \sigma_i(v)$ for some n, i, where

$$\sigma_k(w) = \sum_{\{m<n: S_m^{(1)}(w)=k\}} S_m^{(2)}(w) \,(\text{mod}\, t), \quad k = 1,\dots,d, \quad w = u, v.$$

We need to show that there exists a many–one routing where two packets would compete for the same link at some step if we use the algorithm induced by T. Consider a many–one routing in which two packets start at (distinct) nodes $u' = (\sigma_1(u),\dots,\sigma_d(u))$ and $v' = (\sigma_1(v),\dots,\sigma_d(v))$ with respective destinations $u' - u$ and $v' - v$. Then by invariance, the first packet moves along the path $p_u + u' - u$ with the type sequence $S(u)$, while the second packet moves along the path $p_v + v' - v$ with the type sequence $S(v)$. After $n - 1$ steps, both arrive at node $\mathbf{0}$. Then at step n, they compete for the same link since $S_n(u) = S_n(v)$. $\square$

Remark. The following stronger condition is often easier to check.

Condition (C1′): *For any two type sequences $S(u)$ and $S(v)$, we have $S_m(u) = S_m(v)$ for all $m < n$* whenever $S_n(u) = S_n(v) \neq (0,0)$ *for some n*

Lemma 3. *A modular DG induces a valid permutation routing algorithm if and only if the following Condition* (C2) *holds.*

Condition (C2) *For any two type sequences $S(u)$ and $S(v)$, we have either*

$$\text{(i)} \quad \sum_{\{m<n:S_m^{(1)}(u)=i\}} S_m^{(2)}(u) = \sum_{\{m<n:S_m^{(1)}(v)=i\}} S_m^{(2)}(v) \,(\mathrm{mod}\, t), \quad \textit{for all } i=1,\dots,d,$$

or

$$\text{(ii)} \quad \sum_{\{m>n:S_m^{(1)}(u)=i\}} S_m^{(2)}(u) = \sum_{\{m>n:S_m^{(1)}(v)=i\}} S_m^{(2)}(v) \,(\mathrm{mod}\, t), \quad \textit{for all } i=1,\dots,d,$$

whenever $S_n(u)=S_n(v)\neq(0,0)$ for some n.

Proof. (*Sufficiency*) Following the sufficiency part of the proof of Lemma 2, suppose two packets compete for the same link at step n, which implies $S_n(u')=S_n(v')\neq(0,0)$, $(u'=u-D(u), v'=v-D(v))$, and the packets are at the same node at the beginning of step n. By Condition(C2), either (i) or (ii) holds, the former leading to the same origin for the packets and the latter leading to the same destination. Both are contradictory to the permutation routing setting.

(*Necessity*) Let T be a modular DG that does not satisfy Condition(C2), i.e. there exist two type sequences $S(u)$ and $S(v)$ such that $S_n(u)=S_n(v)\neq(0,0)$, $\sigma_i(u)\neq\sigma_i(v)$, $\tau_{i'}(u)\neq\tau_{i'}(v)$ for some n,i,i', where for $w=u,v,\ k=1,\dots,d$,

$$\sigma_k(w) = \sum_{\{m<n:S_m^{(1)}(w)=k\}} S_m^{(2)}(w),\ \tau_k(w) = \sum_{\{m\geqslant n:S_m^{(1)}(w)=k\}} S_m^{(2)}(w).$$

Consider a permutation routing in which two packets start at (distinct) nodes

$$u' = (\sigma_1(u),\dots,\sigma_d(u)), \quad v' = (\sigma_1(v),\dots,\sigma_d(v))$$

with (distinct) destinations $u''=(-\tau_1(u),\dots,-\tau_d(u)), v''=(-\tau_1(v),\dots,-\tau_d(v))$. Note that the kth digit of $u'-u''$ equals $\sum_{\{m:S_m^{(1)}(u)=k\}} S_m^{(2)}(u)$ which is the kth digit of u. Thus $u'-u''=u$, and similarly $v'-v''=v$. Based on T, the packets move along the paths p_u+u'' and p_v+v''. After $n-1$ steps, both arrive at node $\mathbf{0}$, and then compete for the same link at step n since $S_n(u)=S_n(v)\neq(0,0)$. This completes the necessity part. □

Remark. The following stronger condition is often easier to verify.

Condition (C2′): *For any two type sequences $S(u)$ and $S(v)$, we have either $S_m(u)=S_m(v)$ for all $m<n$ or $S_m(u)=S_m(v)$ for all $m>n$ whenever $S_n(u)=S_n(v)\neq(0,0)$ for some n.*

A modular DG is called a *many–one* (*permutation*) modular DG if it satisfies Condition (C1) (Condition (C2)). Obviously, a many–one modular DG is also a permutation modular DG.

Theorem 1. *Let $T(T')$ be a many–one modular DG for the t-nary $d(d')$-cube. Then T and T' combined yields an invariant oblivious algorithm with $l(T)+l(T')$ steps for permutation routing in the $(d+d')$-cube.*

Proof. We consider the following two-phase algorithm. For phase 1, packets move along the first d dimensions using T (as a many–one routing) while in phase 2, packets move along the last d' dimensions using the reverse of T' (as a one–many routing). More precisely, in phase 1, the $t^{d+d'}$ nodes are divided into $t^{d'}$ subsets of size t^d where the nodes in a subset all have the same last d' digits. As each subset is isomorphic to the d-cube, routing packets in a subset is equivalent to many–one routing in the d-cube, which can be done in $l(T)$ steps by using T. In phase 2, the nodes are divided into subsets of size $t^{d'}$ where the nodes in a subset all have the same first d digits. Since routing packets in each of these subsets is equivalent to one–many routing in the d'-cube, it can be done in $l(T')$ steps by using the reverse of T'. □

Remark. Consider a situation where each node contains a sequence of packets to be transmitted to other nodes. Suppose the first packets of all nodes have distinct destinations, and so do the second (third, etc.) packets. In other words, a sequence of permutation routings is to be conducted. Then the idea of combining T and T' in Theorem 1 can be modified so as to transmit packets two at a time as follows. In phase 1, transmit the first packets along the first d dimensions using T while moving the second packets along the last d' dimensions using T'. In phase 2, transmit the first packets along the last d' dimensions using the reverse of T' while moving the second packets along the first d dimensions using the reverse of T. Thus, two packets per node are successfully transmitted in $2\max\{l(T), l(T')\}$ steps. If $d = d'$ and $T = T'$, this will save the number of steps per packet by a factor of 2.

3. Oblivious algorithms for small d

The following theorem provides invariant oblivious algorithms for binary d-cubes with $d \leqslant 12$.

Theorem 2. *There exists a* $7(8, 11, 14, 19, 24)$*-step invariant oblivious minimum-routing algorithm for permutation routing in the binary* $7(8, 9, 10, 11, 12)$*-cube.*

Proof. By Theorem 1, it suffices to construct a 3(4,7,12)-step many–one modular DG for the binary 3(4,5,6)-cube. Figs. 3–6 present such modular DGs. In these figures, "•" indicates staying put, e.g. in the path $(0,1,0,1) \to (0,1,0,0) \to \bullet \to (0,1,0,0) \to (0,0,0,0)$, "•" is interpreted as $(0,1,0,0)$. Also, if several vertices are labeled by the same node u, the path p_u is determined by the vertex closest to the root. In the following discussion, we shall abuse notation by writing $S_n(u)$ for $S_n^{(1)}(u)$ since the second component of $S_n(u)$ equals 1 (unless $S_n(u)$ is type-0). It is easily verified that the modular DGs for $d = 3, 4, 5$ satisfy Condition (C1′) by noting:

(i) $S_n(u) = S_n(v) \neq 0$ never occurs when there is an edge of type 0 just before the nth edge in the path p_u (or p_v);

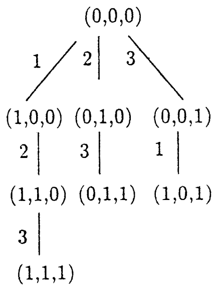
Fig. 3. A modular DG for the binary 3-cube.

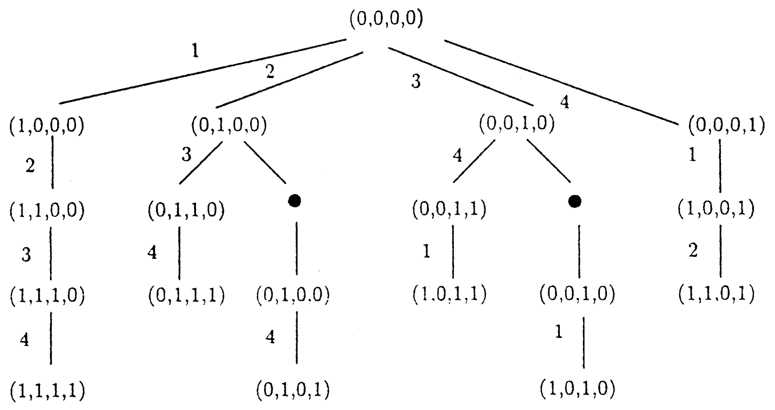
Fig. 4. A modular DG for the binary 4-cube.

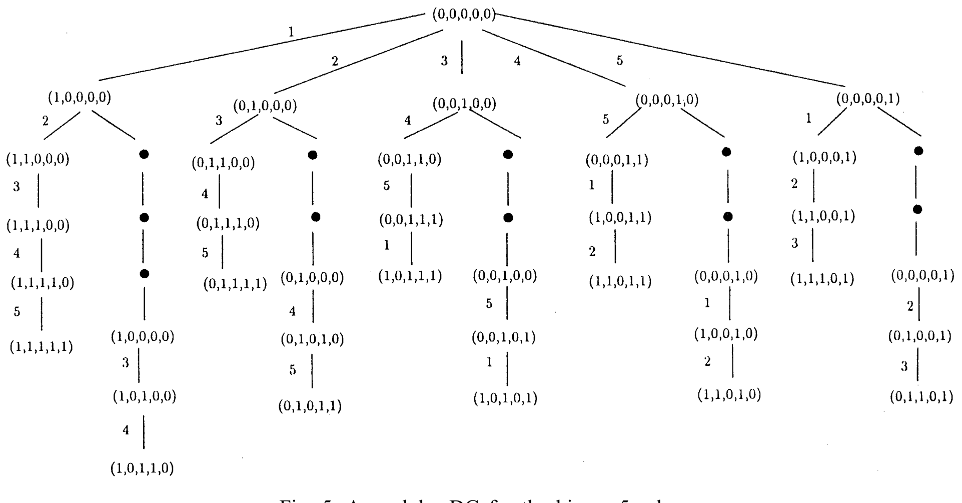
Fig. 5. A modular DG for the binary 5-cube.

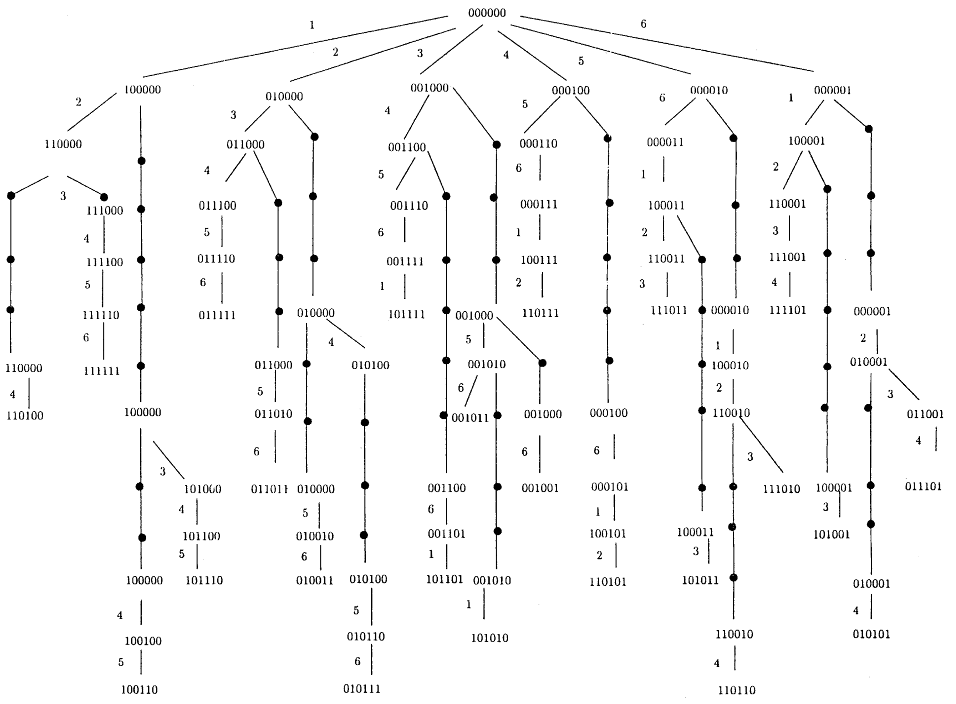

Fig. 6. A modular DG for the binary 6-cube.

(ii) All paths (with no edges of type 0) have cyclic type sequences, i.e. type i is followed by type $i-1 \pmod d$. Hence, $S_n(u)=S_n(v)\neq 0$ forces $S_m(u)=S_m(v)$ for $m<n$.

Due to the presence of many type-0 edges, it is more involved to check that the modular DG for $d=6$ satisfies Condition (C1′). For any two paths containing no type-0 edge, Condition (C1′) is easily verified since the corresponding type sequences are cyclic. Also, note that except for $u=(1,1,1,1,1,1)$, $|p_u|\leqslant 5$ if p_u contains no type-0 edge. On the other hand, if a path includes some type-0 edge(s), the type-0 edges form one or two strings each of length at least 4, and the non-zero-type edges are divided into two or three segments each of length at most 3. Thus, if p_u contains no type-0 edge and p_v contains at least one type-0 edge, then "$S_n(u)=S_n(v)\neq 0$" can occur only when $n\leqslant 3$, which would force $S_m(u)=S_m(v)$ for all $m<n$ due to the cyclic property.

It remains to consider the case that p_u and p_v each contain some (in fact at least 4) type-0 edges. For each $i=1,\dots,6$, consider all paths including one edge of type i (after some type-0 edges). In most cases, this type-i edge appears in different positions for different paths. For example, for $i=3$, among all paths containing a type-3 edge (after some type-0 edges), there are only two pairs of paths ($\{p_{(011010)}, p_{(001010)}\}, \{p_{(011011)}, p_{(001011)}\}$) such that the two paths in the first (second) pair have the 6th (7th) edge being of type 3. For these two pairs, Condition (C1′) is satisfied. □

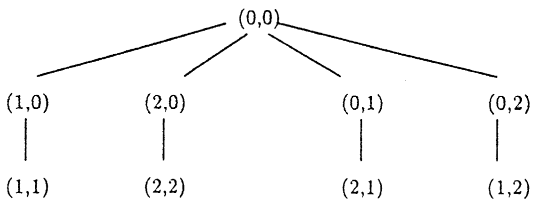

Fig. 7. A modular DG for $t=3,\ d=2$.

Remark. The modular DG in Fig. 5 attains the lower bound $\lceil (2^5-1)/5(2-1)\rceil = 7$ given in Lemma 1, so that it cannot be improved. The modular DG in Fig. 6 misses the lower bound (11) by 1. It is of interest to know whether the lower bound for this case can be attained by an invariant oblivious many–one algorithm. In the many–one routing setting, for $d \leqslant 6$, our algorithm requires fewer steps than that of Kaklamanis et al. [6]. Valiant and Brebner [9] introduced a random (permutation) routing in binary d-cubes with $2d$ steps which succeeds with high probability. By Theorem 2, our (deterministic) algorithm requires fewer steps (except for $d=12$). On the other hand, the permutation routing algorithm of Kaklamanis et al. attains the optimal order as $d \to \infty$. Again, our (small d) algorithm requires fewer steps in the permutation routing setting. (They also proposed to save a factor of two in the time complexity by dividing the packets into two halves where the first half uses one of the two edge-disjoint partitions in the many–one routing phase, while the second half uses the second; and similarly for the one–many routing phase. However, this is shown in [4] to be impossible.) Finally, it may be worth noting that, by the lower bound of Theorem 3.23 in [7] (which is a slight modification of Theorem 1 in [6]), tight oblivious permutation-routing algorithms for binary d-cubes cannot exist for $d \geqslant 19$.

For the rest of this section, we give partial results on the t-nary d-cubes with $d \leqslant 4$.

Lemma 4. *There exists a $\lceil (t+1)/2\rceil$-step invariant oblivious minimum-routing algorithm for many–one routing in the t-nary 2-cube ($t \geqslant 2$).*

Proof. Consider the following modular DG. The path for node $(i,0)$ (node $(0,j)$) is simply $(i,0) \to (0,0)$ ($(0,j) \to (0,0)$). Let $A \equiv \{1,\dots,\lceil (t-1)/2\rceil\}$ and $B \equiv \{\lceil (t+1)/2\rceil,\dots,t-1\}$. For $i,j \in A$, the path for node (i,j) is $(i,j) \to (i,0)_j \to (0,0)$ where $(i,0)_j \equiv (i,0) \to (i,0) \to \cdots \to (i,0)$ ($j(i,0)'s$); for $i,j \in B$, the path for node (i,j) is $(i,j) \to (i,0)_{j-\lceil (t-1)/2\rceil} \to (0,0)$; for $i \in A,\ j \in B$, the path for node (i,j) is $(i,j) \to (0,j)_i \to (0,0)$; for $i \in B,\ j \in A$, the path for node (i,j) is $(i,j) \to (0,j)_{i-\lceil (t-1)/2\rceil} \to (0,0)$. It is easily checked that Condition (C1′) is met. (Fig. 7 plots the modular DG for $t=3$.)

The algorithm given in Lemma 4 attains the lower bound of Lemma 1, since $\lceil (t^d-1)/d(t-1)\rceil = \lceil (t+1)/2\rceil$. In particular, for $t=3$, we have a tight invariant oblivious minimum-routing algorithm for many–one routing in the ternary 2-cube. By Theorem 1, we have a tight invariant oblivious minimum-routing algorithm for permutation routing in the ternary 4-cube. This algorithm based on the DG method is much simpler than that given in Hwang et al. [5].

Our final result concerns the existence of a tight invariant oblivious algorithm for permutation routing in the t-nary 3-cube. While such an algorithm can be readily constructed for $t=2,3$, we show in Theorem 3 below that there exists no such algorithm for $t \geqslant 4$. (It is worth noting that as a consequence of Theorem 3.23 in [7], there exists no tight oblivious algorithm for $t \geqslant 322$.)

Theorem 3. *There exists no tight invariant oblivious algorithm for permutation routing in the t-nary 3-cube with* $t \geqslant 4$.

Proof. Suppose there is a 3-step invariant oblivious algorithm for permutation routing with its modular DG denoted by T. Necessarily, all the vertices labeled by nodes (i,j,k) with $i,j,k \neq 0$ are level-3 (i.e. leaves of the tree). Let X denote the set of these vertices so that $|X| = (t-1)^3$. Let E_2 denote the set of level-2 edges below which there is at least one (level-3) vertex in X. (Thus, none of the edges in E_2 can be of type 0.) For each $e \in E_2$, let $X_e = \{x \in X : x \text{ is below } e\}$. Denote the type of an edge e by $\tau(e)$. For a level-2 edge $e \in E_2$ which is below a level-1 edge e', if $\tau(e) = (i,j)$ and $\tau(e') = (i',j')$, then every vertex in X_e must have a label with the ith and i'th digits being j and j'. Consequently, for $e \in E_2$, $1 \leqslant |X_e| \leqslant t-1$.

Claim 1. *For* $x_3, y_3 \in X$, *the following situation cannot happen*:

$$x_3 \xrightarrow{e_3} x_2 \xrightarrow{e_2} x_1 \xrightarrow{e_1} \mathbf{0}, \quad y_3 \xrightarrow{f_3} y_2 \xrightarrow{f_2} y_1 \xrightarrow{f_1} \mathbf{0},$$

and $\tau(e_2) = \tau(f_2), \tau(e_1) \neq \tau(f_1), \tau(e_3) \neq \tau(f_3)$ *where* $x_3 \xrightarrow{e_3} x_2$ *signifies level-3 edge* e_3 *connecting level-3 vertex* x_3 *with level-2 vertex* x_2.

Proof. This follows from the necessity of Condition (C2) in Lemma 3.

For $i = 1,2,3, \quad j = 1,\ldots,t-1, \quad$ let $E_2(i,j) = \{e \in E_2 \colon \tau(e) = (i,j)\}$.

Claim 2. *If* $E_2(i,j)$ *contains an* e_2 *with* $|X_{e_2}| \geqslant 2$, *then* $|E_2(i,j)| = 1$.

Proof. Suppose to the contrary that $E_2(i,j)$ contains another edge f_2. Let $x_3, x_3' \in X_{e_2}$, $y_3 \in X_{f_2}$ with respective paths

$$x_3 \xrightarrow{e_3} x_2 \xrightarrow{e_2} x_1 \xrightarrow{e_1} \mathbf{0}, \quad x_3' \xrightarrow{e_3'} x_2 \xrightarrow{e_2} x_1 \xrightarrow{e_1} \mathbf{0},$$

$$y_3 \xrightarrow{f_3} y_2 \xrightarrow{f_2} y_1 \xrightarrow{f_1} \mathbf{0}.$$

Since $e_2 \neq f_2$ and $\tau(e_2) = \tau(f_2) = (i,j)$, we must have $\tau(e_1) \neq \tau(f_1)$ (otherwise $x_1 = y_1$, $x_2 = y_2$, and e_1 and e_2 would be merged with f_1 and f_2). On the other hand, $\tau(f_3)$ must be different from one of $\tau(e_3)$ and $\tau(e_3')$ since the latter two cannot be the same. By Claim 1, this leads to a contradiction, thereby proving Claim 2.

Claim 3. *If* $|X_e| = 1$ *for all* $e \in E_2(i,j)$, *then* $|E_2(i,j)| \leqslant t-1$.

Proof. For any two different $e_2, f_2 \in E_2(i,j)$, let $e_1(f_1)$ be the edge above $e_2(f_2)$, and $e_3(f_3)$ the level-3 edge below $e_2(f_2)$ that connects the only vertex in $X_{e_2}(X_{f_2})$. Since $\tau(e_1) \neq \tau(f_1)$, we must have $\tau(e_3) = \tau(f_3)$ (otherwise a contradiction would arise by Claim 1). Suppose the first component of $\tau(e_3) = \tau(f_3)$ equals $i'(\neq i)$. Then the first component of $\tau(e_1)$ (and $\tau(f_1)$) must be the only value in $\{1,2,3\}$ other than i and i'. Hence, there are only $t-1$ possible choices for $\tau(e_1)$, establishing the claim.

By Claims 2 and 3 together with the fact that $|X_e| \leqslant t-1$, we have

$$\left| \bigcup_{e \in E_2(i,j)} X_e \right| \leqslant t-1 \quad \text{for each } i,j,$$

so that

$$|X| = \left| \bigcup_{e \in, E_2} X_e \right| \leqslant 3(t-1)^2.$$

This contradicts $|X| = (t-1)^3$ if $t \geqslant 5$.

It remains to consider the case $t = 4$. Let $X(i,j) = \bigcup_{e \in E_2(i,j)} X_e$, for $i,j = 1,2,3$. Since $|X(i,j)| \leqslant t-1 = 3$ and since

$$\left| \bigcup_{i,j=1,2,3} X(i,j) \right| = |X| = 27,$$

it follows that the nine sets $X(i,j), i,j = 1,2,3$ are disjoint each having three elements. Denote by $[a,b,\times]$ the set $\{(a,b,c) : c = 1,2,3\}$ (similarly for $[a,\times,b]$ and $[\times,a,b]$). Letting $U(i,j) = \{u_x^* : x \in X(i,j)\}$, it follows from the proofs of Claims 2 and 3 that $U(1,j)$ must be one of $[j,k,\times]$, $[j,\times,k], k = 1,2,3$ (similarly for $U(2,j)$ and $U(3,j)$). We may assume, without loss of generality, that $U(1,1) = [1,1,\times]$. Then $U(2,1)$ must be either $[2,1,\times]$ or $[3,1,\times]$, since $U(1,1) \cup U(2,1) = \phi$. Without loss of generality, assume $U(2,1) = [2,1,\times]$. Then $U(1,2)$ is either $[2,2,\times]$ or $[2,3,\times]$. Again, without loss of generality, assume $U(1,2) = [2,2,\times]$. Then $U(2,2)$ is either $[1,2,\times]$ or $[3,2,\times]$. We need to consider these two cases separately.

Case (i): $U(2,2) = [1,2,\times]$. We have $U(3,1) = [3,\times,1]$ or $[\times,3,1]$. If $U(3,1) = [3,\times,1]$, then $U(1,3) = [3,\times,2]$ or $[3,\times,3]$, the former resulting in no choice for $U(3,2)$ and the latter resulting in no choice for $U(3,3)$. If $U(3,1) = [\times,3,1]$, then $U(2,3) = [\times,3,2]$ or $[\times,3,3]$, the former resulting in no choice for $U(3,2)$ and the latter resulting in no choice for $U(3,3)$.

Case (ii): $U(2,2) = [3,2,\times]$. We have $U(1,3) = [3,1,\times]$ or $[3,3,\times]$, the latter resulting in no choice for $U(3,1)$. Thus, $U(1,3) = [3,1,\times]$, implying $U(3,1) = [\times,3,1]$ and $U(3,2) = [\times,3,2]$ and $U(3,3) = [\times,3,3]$. Then there is no choice for $U(2,3)$. This completes the proof for $t = 4$. $\square$

References

[1] A. Borodin, J.E. Hopcroft, Routing, merging and sorting on parallel models of computation, J. Comput. Syst. Sci. 30 (1985) 130–145.

[2] A. Borodin, P. Raghavan, B. Schieber, E. Upfal, How much can hardware help routing, Proceedings of the 25th ACM Symposium Theory Computing, 1993, pp. 573–582.

[3] M.D. Grammatikakis, D.F. Hsu and F.K. Hwang, Universality of d-cube ($d < 8$), in: D.G. Joubert, D. Trystram, F.J. Peters (Eds.), Proceedings of the Advancement in Parallel Computing, Elsevier, Amsterdam, 1993.

[4] F.K. Hwang, Y.C. Yao, Comments on the oblivious routing algorithm of Kaklamanis, Krizanc and Tsantilas in the hypercube, Theory of Computing Systems 31 (1998) 63–66.

[5] F.K. Hwang, Y.C. Yao, M.D. Grammatikakis, A d-move local permutation routing for the d-cube, Discrete Appl. Math. 72 (1997) 199–207.

[6] C. Kaklamanis, D. Krizanc, T. Tsantilas, Tight bounds for oblivious routing in the hypercube, Math. Syst. Theory 24 (1991) 223–232.

[7] F.T. Leighton, Introduction to Parallel Algorithms and Architectures, Morgan Kaufmann Publishers, San Mateo, Ca, 1992.

[8] L.G. Valiant, A scheme for fast parallel communication, SIAM J. Comp. 11 (1982) 350–361.

[9] L.G. Valiant, G.J. Brebner, Universal schemes for parallel communication, STOC (1981) 263–277.