An Algorithm for Generating Node Disjoint Routes in Kautz Digraphs

Gerard J.M. Smit, Paul J.M. Havinga, Pierre G. Jansen

University of Twente Dept. of Computer Science
P.O. Box 217, 7500 AE Enschede
the Netherlands

Abstract

In this paper we focus on a particular class of interconnection networks: Kautz networks [9]. These networks have nice properties: a network with degree d and $N = d^k + d^{k-1}$ nodes (for any cardinal $d, k > 0$), has a diameter of at most $d \log N$, the degree d is fixed and independent of the network size. The network is fault-tolerant and the connectivity is d . There is a straightforward mapping from standard computation graphs such as a linear array, a ring and a tree to a Kautz network.

The network allows for a simple routing algorithm, even when nodes or links are faulty. There exists d node disjoint paths between any pair of vertices [8]. In this paper we present an algorithm to generate these node disjoint routes. The routes are delivered with increasing length and are free of loops. We prove that these routes are node disjoint and as short as possible.

Keywords: interconnection networks, Kautz graphs, routing algorithms, routing hardware.

1. Introduction

COMPAS (COMMunication in PARallel Systems) is a research project of the University of Twente. One of the goals of this project is the design, realization and evaluation of interconnection networks for parallel systems.

In this paper we restrict to regular interconnection networks for multi-computer systems. We define a multi-computer system as a collection of linked node computers (abbreviated as nodes), in which the nodes communicate via message passing [5]. We assume that each node consists of a communication processor, a node memory, and one or more node processors.

We start with some definitions from graph theory. See [1] for more definitions. A communication network is modelled as a *graph* $G = (V, E)$ or a digraph in which the vertices represent the nodes. Communication links are represented by edges (bi-directional links) or arcs (uni-directional links). Let $N(x)$ denote the set of neighbours to a vertex x . The *degree* $d(x)$ is the cardinality of $N(x)$. A graph is called *degree-regular* if all vertices have the same degree. Let $in_N(x)$ be the set of neighbours that have an arc pointing

to x . The *in_degree* $in(x)$ is the cardinality of $in_N(x)$. The *out_degree* $out(x)$ can be defined in a similar way.

A *path* (also called *route*) $R(x, y)$ is a sequence of vertices such that the first vertex of $R(x, y)$ is x and the last vertex is y and each vertex is incident to the two vertices immediately preceding and following it. The *distance* $dist(x, y)$ between vertices x and y is the length of the shortest path between x and y . The *diameter* $diam(G)$ is the maximum distance over all pairs of vertices. A graph G is *k-connected* if there exists k internally vertex disjoint paths between any pair of vertices. The *connectivity* $K(G)$ is the largest integer k such that G is k -connected.

In this paper we focus on the so called 'Kautz network', after its inventor W.H. Kautz [9]. This network has drawn some attention recently [2], [6], [8]. Section 2 defines the Kautz network and summarizes some interesting properties of this network. In Section 3 a new algorithm for finding the d node disjoint paths between two arbitrary nodes is presented. Unity [4] is used as a description language for the algorithm. In Section 4 we prove that these routes are node disjoint.

2. Definition and properties of Kautz graphs

The Kautz graph was first defined by W.H. Kautz in [9]. Recently the Kautz graph has been re-discovered by Imase et al. [7] and by Fiol et al. [6]. Imase and Itoh gave an *arithmetic definition*. Fiol defined the Kautz graph in terms of a *line digraph iteration*. The Kautz graph has some similarity with the well known de Bruijn graph [3].

In this paper we will focus on directed Kautz graphs (Kautz digraphs). The undirected Kautz graphs can be obtained from the associated digraphs by removing the orientation, and the parallel edges. Throughout this paper we will use the definition of Kautz.

Definition of Kautz graphs [9]:

The Kautz digraph $K(d, k)$ with *in_degree* and *out_degree* d and diameter k is the digraph whose vertices are labeled with words of length k ($x_1, \dots, x_k$) from an alphabet of $d+1$ letters by removing those words in which there are two consecutive identical letters ($x_i \neq x_{i+1}$, for $1 \leq i \leq k-1$). There is an arc from a vertex x to a vertex y if and only if the last

k-1 letters of x are the same as the first k-1 letters of y . In other words the vertex $(x_1, \dots, x_k)$ is neighbor to d vertices $(x_2, \dots, x_k, z)$, where z is any letter different from x_k .

Properties of Kautz digraphs

Some properties of Kautz digraphs are:

- The number of vertices $N = d^k + d^{k-1}$.

Degree d	Diameter k	Number of vertices of $K(d,k)$
2	8	384
3	8	8.748
4	8	81.920
5	8	468.750

Table 1: Number of vertices of some Kautz graphs.

- The graph is *degree-regular*. The degree is fixed (in_degree and out_degree d ; for all $k, d > 0$). Networks of arbitrary large size can be built using (VLSI) components as nodes with a *fixed* number of connections per node. Other networks, such as the binary cube, require the number of connections per node to *increase* with the number of nodes.

Fig. 1: Example of a Kautz graph $K(2,3)$.

- The *diameter* of the network is at most k ($< d \log N$).
- A Kautz network is *fault tolerant*. Imase et al. [8] showed the existence of d vertex disjoint paths between any pair of vertices in $K(d,k)$, one of a length of at most k , $d-3$ of a length of at most $k+1$ and two of a length of at most $k+2$. This means that the performance degradation due to an increase in routing distances resulting from faults may be fairly low.
- Another interesting property of the network is the fact that it admits *self routing* of messages, both when the network is fault free as well as when some nodes or links become faulty (see section 3).
- A Kautz graph can *emulate standard computation graphs*. Fiol [6] showed that Kautz digraphs are line digraphs of Eulerian digraphs, so a Kautz graph contains a Hamilton cycle. This implies that a linear array and a ring can be mapped on it. Furthermore a tree can be

mapped on a $K(d,k)$ graph. Because the diameter of a $K(d,k)$ is at most $d \log N$ a mesh can be embedded in $O(d \log N)$.

3. Routing in a Kautz graph.

3.1 Notations and definitions.

Let $K = (V, E)$ be a Kautz digraph, and $x = (a_1, \dots, a_k)$ and $y = (b_1, \dots, b_k)$ denote two arbitrary chosen nodes from K . V , respectively source and destination node.

A *route* $R(x, y)$ is defined as a sequence $C = \langle c_1, \dots, c_1, \dots, c_n \rangle$ of letters, in which $c_i \in \{0, \dots, d\}$ for all $0 < i \leq n$ and $c_i \neq c_{i+1}$ for all $0 < i < n$. The first k letters of C , $\langle c_1, \dots, c_k \rangle$, are equal to x and the last k letters of C is $\langle c_{n-k+1}, \dots, c_n \rangle$ are equal to y . In this definition each sequence of k contiguous letters of C denote an *intermediate node* on the route from x to y . In particular the node whose sequence of k letters starts on the second letter of $R(x, y)$ is called the *out_neighbour* on_R of x . The node whose sequence of k letters ends on the one but last letter of $R(x, y)$ is called the *in_neighbour* in_R of y . We define the set *out_neighbours* $ON(x)$ as the set of all *out_neighbours* of x , i.e. the set of nodes that have an arc coming from node x .

$$ON(x) = \{ z : z \in \{0 \dots d\} \wedge z \neq a_k :: (a_2, \dots, a_k, z) \}$$

Note that because $z \neq a_k$, K does not contain selfloops. In analogy we can define the set *in_neighbours* $IN(y)$ as:

$$IN(y) = \{ z : z \in \{0 \dots d\} \wedge z \neq b_1 :: (z, b_1, \dots, b_{k-1}) \}$$

Example:

In figure 1 $Re = \langle 021201 \rangle$ is a route from (021) to (201) via node (212) and (120). $on_{Re} = (212)$; $in_{Re} = (120)$; $ON(021) = \{(212), (210)\}$; $IN(201) = \{(120), (020)\}$.

3.2 Generic routing in Kautz graphs.

To find a route $R(x, y)$ from x to y , we select one of the *out_neighbour* vertices of x . In particular we can select $(a_2, \dots, a_k, b_1)$ as the first intermediate node (provided $b_1 \neq a_k$). By applying a 'shift' operation on the letters of the source $x = (a_1, \dots, a_k)$, where a_1 is shifted out and b_1 is shifted in, we find the first intermediate node $(a_2, \dots, a_k, b_1)$ of the route $R(x, y)$ from x to y (see Table 2). This process of shifting out source

$(a_1, a_2, a_3, \dots, a_{k-1}, a_k)$	(source node)
$(a_2, a_3, \dots, a_{k-1}, a_k, b_1)$	(first node: $\in ON(x)$)
$(a_3, \dots, a_{k-1}, a_k, b_1, b_2)$	(second intermediate node)
$\dots$	
$\dots$	
$(a_k, b_1, b_2, \dots, b_{k-1})$	(last node: $\in IN(y)$)
$(b_1, b_2, \dots, b_{k-1}, b_k)$	(destination node)

Table 2: Generic routing
(assume $a_k \neq b_1$).

letters and shifting in destination letters continues until the source word is completely replaced by the destination word. In this way we can (by simple concatenation of source and destination word) generate a straightforward generic route $R(x,y) = \langle a_1, \dots, a_k, b_1, \dots, b_k \rangle$ from source to destination with a length of k . (The case $b_1 = a_k$ is covered in the next section.)

3.3 Generating node disjoint routes.

Imase et al. [8] showed the existence of d vertex disjoint paths between any pair of vertices in K . In their approach they assumed that there exists only one route of a length of at most k . However in general there may be more than one route with length $\leq k$. Moreover, a route with the Imase approach might contain loops.

Example:

Using the graph of figure 1 we find the following two routes from (120) to (201) with the Imase approach:

$R_1 = \langle 1201 \rangle$ and $R_2 = \langle 12020201 \rangle$. $\{R_2$ contains a loop!}

Our algorithm finds the following two routes:

$R_3 = \langle 1201 \rangle$ and $R_4 = \langle 120201 \rangle$. $\{R_3$ and R_4 both have a length $\leq k\}$

Our routing mechanism does not lead to loops in routes. The loops created with the Imase algorithm can be eliminated with a relatively simple method, but unfortunately sometimes backtracking is needed. Backtracking will lead to a complex hardware implementation of the routing algorithm.

In this section we describe a new algorithm that generates routes, without backtracking, that are as short as possible and free of loops. We first give an informal description of the algorithm. After that we give a specification in Unity [4]. In the next section we prove that the routes generated with our algorithm are free of loops and node disjoint.

3.3.1 Informal description.

Each node x has d out_neighbours ($|ON(x)| = d$) and each node y has d in_neighbours ($IN(y)$). The first intermediate node of each route is a particular element of $ON(x)$. Similarly the last intermediate node of each route is an element of $IN(y)$. To find node disjoint routes between x and y we connect the nodes of set $ON(x)$ with the nodes of set $IN(y)$ in a node disjoint fashion. Fig. 2 shows the set of nodes $ON(x)$ connected to x and the set of nodes $IN(y)$ connected to y .

First we define three sets. The set *Found_Routes* FR denote the set of routes that have been found so far.

For all routes from FR we can define the set *used_out_neighbours* $ON_u(x)$ as a set of out_neighbours of the routes from set FR. This means that if a node m is an element of $ON_u(x)$ then there is a route R_m ($R_m \in FR$) such that $m =$

Fig. 2.: Routing strategy.

on_{R_m} . Similarly let the set *used_in_neighbours* $IN_u(y)$ denote the set of in_neighbours of routes from FR. Initially the sets $ON_u(x)$, $IN_u(y)$ and FR are empty. Note that $ON_u(x) \subseteq ON(x)$ and $IN_u(y) \subseteq IN(y)$.

Informally the *strategy* to find the node disjoint routes is:

Generate routes $R_i(x,y)$ between x and y with increasing length starting from the shortest route. Add this route to the set of found routes FR only if the out_neighbour node and in_neighbour node of this route are not element of $ON_u(x)$ respectively $IN_u(y)$. Subsequent add the neighbour nodes on_{R_i} to the set $ON_u(x)$ and in_{R_i} to $IN_u(y)$, to indicate that they are used now. Continue until all d routes are generated.

The properties of the set of routes FR are:

1. There are exactly d routes in FR.
2. The routes are generated with increasing length.
3. The routes of FR are node disjoint (Theorem).
4. The maximum length of the routes is $k+2$.
5. The routes are free of loops.

The proofs of these properties are given in section 4.

3.3.2 The algorithm in short.

We divide the set of routes in three classes according to the length:

- Class I: Routes with length $\leq k$ (R_I).
- Class II: Routes with length $k+1$ (R_{II}).
- Class III: Routes with length $k+2$ (R_{III}).

A. Routes class I.

We first define a Boolean function *overlap* (v,x,y) on x and y which is True if the last v letters of x equals the first v letters of y , and is False otherwise. If $v = 0$ the function returns True if $b_1 \neq a_k$.

$$\text{overlap}(v,x,y) = \{(\bigwedge i : 1 \leq i \leq v :: a_{k-v+1} = b_i) \vee (v = 0 \wedge b_1 \neq a_k)\}$$

Because x and y have unique values, we can define a unique, ordered set V of integers as:

$$V = \{v : 0 \leq v \leq k \wedge \text{overlap}(v, x, y) :: v\}.$$

There is a unique, ordered set of routes with length $\leq k$. It is obvious that there are no routes of length $\leq k$ that have the same length.

By applying function *overlap*(v, x, y) on x and y , with decreasing v starting from $v = k$, we find all routes R_i with length $0 < v \leq k$.

Example:

If $x = (a_1, \dots, a_{k-v}, a_{k-v+1}, \dots, a_k)$ and $y = (b_1, \dots, b_v, b_{v+1}, \dots, b_k)$ and $(a_{k-v+1}, \dots, a_k) = (b_1, \dots, b_v)$, then the route is defined by the sequence: $\langle a_1, \dots, a_{k-v}, b_1, \dots, b_k \rangle = \langle a_1, \dots, a_k, b_{v+1}, \dots, b_k \rangle$.

The set of routes R_i is uniquely defined by the set of sequences:

$$R_i = \{v : v \in V \wedge (a_2, \dots, a_k, b_{v+1}) \notin \text{ON}_u(x) \wedge (a_{k-v}, b_1, \dots, b_{k-1}) \notin \text{IN}_u(y) :: \langle a_1, \dots, a_k, b_{v+1}, \dots, b_k \rangle = \langle a_1, \dots, a_{k-v}, b_1, \dots, b_k \rangle\}$$

So when $\text{on}_{R_i} \notin \text{ON}_u(x)$ and $\text{in}_{R_i} \notin \text{IN}_u(y)$ then on_{R_i} and in_{R_i} are added to $\text{ON}_u(x)$ and $\text{IN}_u(y)$ respectively.

B. Routes of class II.

If all routes with length $\leq k$ have been generated, the routes of class II (R_{II}) can be found by inspecting the set $\text{ON}(x)$ and $\text{IN}(y)$. If the set $\text{ON}(x)$ contain nodes $c_{\text{ON}(x)}$ such that the last letter of $c_{\text{ON}(x)}$ equals the first letter of a node $c_{\text{IN}(y)}$ of $\text{IN}(y)$ and for both nodes $c_{\text{ON}(x)} \notin \text{ON}_u(x)$ and $c_{\text{IN}(y)} \notin \text{IN}_u(y)$, then we can add a route of length $k+1$ with $c_{\text{ON}(x)}$ as the out_neighbour node and $c_{\text{IN}(y)}$ as the in_neighbour node to the set of routes FR.

The set of routes R_{II} is defined as a set:

$$R_{II} = \{c_1 \in \text{ON}(x) \wedge c_1 \in \text{IN}(y) \wedge (a_2, \dots, a_k, c_1) \notin \text{ON}_u(x) \wedge (c_1, b_1, \dots, b_{k-1}) \notin \text{IN}_u(y) :: \langle a_1, \dots, a_k, c_1, b_1, \dots, b_k \rangle\}.$$

C. Routes of class III.

Routes of length $k+2$ can be found by arbitrary choosing one node u of set $\text{ON}(x)$ with $u \notin \text{ON}_u(x)$ as first intermediate node and choosing one node v from $\text{IN}(y)$ with $v \notin \text{IN}_u(y)$ as the last intermediate node.

$$R_{III} = \{c_2 \in \text{ON}(x) \wedge c_3 \in \text{IN}(y) \wedge c_2 \neq c_3 \wedge (a_2, \dots, a_k, c_2) \notin \text{ON}_u(x) \wedge (c_3, b_1, \dots, b_{k-1}) \notin \text{IN}_u(y) :: \langle a_1, \dots, a_k, c_2, c_3, b_1, \dots, b_k \rangle\}.$$

Program 1 gives a formal specification of the algorithm to generate routes.

4. Proof that all routes of FR are node disjoint.

Due to lack in space we can only show the outline of the proof. The complete proofs can be found in [10].

Theorem: All routes of the set FR are node disjoint.

Program generate_routes

Initially FR, $\text{ON}_u(x)$, $\text{IN}_u(y)$, $v, i := \emptyset, \emptyset, \emptyset, 0, 0$
always

Route_class_I =
 $\text{overlap}(k-v, x, y) \wedge$
 $(a_2, \dots, a_k, b_{v+1}) \notin \text{ON}_u(x) \wedge$
 $(a_{k-v}, b_1, \dots, b_{k-1}) \notin \text{IN}_u(y)$
 $R_i = \langle a_1, \dots, a_k, b_{v+1}, \dots, b_k \rangle$

Route_class_II =
 $(a_2, \dots, a_k, i) \notin \text{ON}_u(x) \wedge$
 $(i, b_1, \dots, b_{k-1}) \notin \text{IN}_u(y) \wedge$
 $(a_k \neq i) \wedge (b_1 \neq i)$
 $R_{II} = \langle a_1, \dots, a_k, i, b_1, \dots, b_k \rangle$

Route_class_III =
 $(a_2, \dots, a_k, i_1) \notin \text{ON}_u(x) \wedge$
 $(i_2, b_1, \dots, b_{k-1}) \notin \text{IN}_u(y) \wedge$
 $(a_k \neq i_1) \wedge (b_1 \neq i_2) \wedge (i_1 \neq i_2)$
 $R_{III} = \langle a_1, \dots, a_k, i_1, i_2, b_1, \dots, b_k \rangle$
 $i_1 = \text{Rand}(\{0..d\}); i_2 = \text{Rand}(\{0..d\})^1$

assign

FR, $\text{ON}_u(x)$, $\text{IN}_u(y)$, $v, i :=$

$(\text{FR} \cup R_i), (\text{ON}_u(x) \cup \text{on}_{R_i}), (\text{IN}_u(y) \cup \text{in}_{R_i}), v+1, 0$
 IF $(v \leq k) \wedge \text{Route_class_I} \sim$
FR, $\text{ON}_u(x)$, $\text{IN}_u(y)$, $v+1, 0$
 IF $(v \leq k) \wedge \text{NOT Route_class_I} \sim$

$(\text{FR} \cup R_{II}), (\text{ON}_u(x) \cup \text{on}_{R_{II}}), (\text{IN}_u(y) \cup \text{in}_{R_{II}}), v, i+1$
 IF $(v = k+1) \wedge (i < d) \wedge \text{Route_class_II} \sim$
FR, $\text{ON}_u(x)$, $\text{IN}_u(y)$, $v, i+1$
 IF $(v = k+1) \wedge (i < d) \wedge \text{NOT Route_class_II} \sim$

FR, $\text{ON}_u(x)$, $\text{IN}_u(y)$, $v+1, 0$
 IF $(v = k+1) \wedge (i = d) \sim$

$(\text{FR} \cup R_{III}), (\text{ON}_u(x) \cup \text{on}_{R_{III}}), (\text{IN}_u(y) \cup \text{in}_{R_{III}}), v, 0$
 IF $(v = k+2) \wedge \text{Route_class_III} \sim$
FR, $\text{ON}_u(x)$, $\text{IN}_u(y)$, $v, 0$
 IF $(v = k+2) \wedge \text{NOT Route_class_III}$

end {generate_routes}

Program 1: generate routes.

1. Rand(S) denotes a function that randomly selects an element from the set S. This is a method for generating all combinations.

We can distinguish six cases to be proved:

- case I: routes of class I are disjoint
- case II: routes of class II are disjoint
- case III: routes of class III are disjoint
- case I/II: routes of class I and class II are disjoint
- case I/III: routes of class I and class III are disjoint
- case II/III: routes of class II and class III are disjoint

First we give two general lemmas without proof:

Lemma S1:

All nodes of a route $R_i(x,y)$ from the set FR are node disjoint (i.e. a route does not contain loops).

Lemma S2:

All routes with equal length from FR are node disjoint.

Because class II and III contain routes of equal length and Lemma S2 states that these routes are disjoint.

From the definition of Kautz graphs it follows that the routes from the set FR must have the following properties:

- *Kautz inequality*: for any letter r_i from a route, $r_i \neq r_{i+1}$ for $1 \leq i < k$.
- *outneighbour inequality*: the out_neighbours of routes from s to d must be unequal.
- *inneighbour inequality*: the in_neighbours of routes from s to d must be unequal.

To clarify the proofs we first show a proof for a special case with two examples. After that we consider the general case.

Special case:**Example A:**

Let $s = (a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$ and $d = (b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8)$ define two nodes of 8 letters. Then $R_I = \langle a_1, \dots, a_8, b_7, b_8 \rangle = \langle a_1, a_2, b_1, \dots, b_8 \rangle$ is a route between s and d of length 2 from class I and $R_{III} = \langle a_1, \dots, a_8, c_2, c_3, b_1, \dots, b_8 \rangle$ is a route between s and d from class III. We prove by contraposition that these two routes are disjoint. We chose a node $n_I = (a_2, \dots, a_8, b_7) = (a_2, b_1, \dots, b_7)$ from R_I and a node $n_{III} = (a_5, \dots, a_8, c_2, c_3, b_1, b_2)$ from R_{III} . Suppose that $n_I = n_{III}$, then we get the following 8 equations (each column is an equation: e.g. $a_3 = b_1 = a_6$):

$$\begin{array}{cccccccc} & & & \xleftrightarrow{g=3} & & & & \\ n_I & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & b_7 \\ & a_2 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\ n_{III} & a_5 & a_6 & a_7 & a_8 & c_2 & c_3 & b_1 & b_2 \\ & & & \xleftarrow{h=5} & & & & & \end{array}$$

In this example we have the following two equivalence classes:

$$\begin{aligned} [\epsilon 1] &= \{c_2, a_6, b_4, a_8, b_6, b_1, a_3, a_5, b_3, a_2\} \\ [\epsilon 2] &= \{c_3, a_7, b_5, a_4, b_2, b_7\}. \end{aligned}$$

We see that for instance a_5 and a_6 are both in $[\epsilon 1]$ which contradicts with the Kautz inequality {end example A}.

Example B:

In this example we use the same routes R_I and R_{III} as in

example A, but now $n_{III} = (a_6, a_7, c_2, c_3, b_1, \dots, b_4)$.

$$\begin{array}{cccccccc} & & & \xleftrightarrow{g=4} & & & & \\ n_I & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & b_7 \\ & a_2 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\ n_{III} & a_6 & a_7 & a_8 & c_2 & c_3 & b_1 & b_2 & b_3 \\ & & & \xleftarrow{h=4} & & & & & \end{array}$$

In this example we have four equivalence classes:

$$\begin{aligned} [\epsilon 1] &= \{b_1, a_3, a_7, b_5\} \\ [\epsilon 2] &= \{b_2, a_4, a_8, b_6\} \\ [\epsilon 3] &= \{b_3, c_2, a_5, b_7\} \\ [\epsilon 4] &= \{b_4, c_3, a_6, a_2\} \end{aligned}$$

Now c_3 is in the same class as a_2 , which is in contradiction with the in_neighbour inequality {end example B}.

These examples show that the assumption $n_I = n_{III}$ induces a repeating pattern in the letters of n_I . The repeating pattern leads to at most g equivalence classes. The number depends on the choice of n_I and n_{III} , i.e. it is the difference between the indices of the first letters of n_I and n_{III} . The equivalence classes are denoted as $[\alpha_i] \dots [\alpha_g]$.

Similarly we have a repetition in the pattern of n_{III} . This leads to h equivalence classes denoted as $[\beta_1] \dots [\beta_h]$, in which h is the difference between the indices of the last letters of n_I and n_{III} .

Example B can be reformulated in terms of equivalence classes. In this matrix we abbreviate the equivalence class $[\alpha_i]$ as α_i . The figure below shows the corresponding *equivalence matrix* in which each column contains elements of one equivalence class. The equivalence matrix shows how the equivalence classes of n_I and n_{III} overlap and are thus related. The equivalence classes overlap in $g+h-2$ columns.

$$\begin{array}{cccccccc} & & & \xleftrightarrow{g=4} & & & & \\ \alpha_4 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_1 & \alpha_2 & \beta_3 \\ \alpha_4 & \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_1 & \beta_2 & \beta_3 \\ & & & \xleftarrow{h=4} & & & & \end{array}$$

The equivalence classes are: $[\epsilon 1] = \{\alpha_1, \beta_1\}$; $[\epsilon 2] = \{\alpha_2, \beta_2\}$; $[\epsilon 3] = \{\alpha_3, \beta_3\}$ and $[\epsilon 4] = \{\alpha_4, \beta_4\}$.

In example A the number equivalence classes is reduced to only two classes, i.e. one corresponding with c_2 and the other with c_3 .

General case.

For deriving the general case we use two more crucial Lemmas [10].

Lemma L1:

If g and h have a common factor (see example B), the assumption $n_I = n_{III}$ leads to $\alpha_g = \beta_h = c_3$, which contradicts with the in_neighbour inequality.

Lemma L2:

If g and h do not have a common factor (see example A), the assumption $n_I = n_{III}$ leads to a violation of the Kautz inequality.

Proof case I/III

Let $\langle a_1, \dots, a_k, b_{i+1}, \dots, b_k \rangle = \langle a_1, \dots, a_{k-i}, b_1, \dots, b_k \rangle$ denote a route of class I and $\langle a_1, \dots, a_k, c_2, c_3, b_1, \dots, b_k \rangle$ denote a route of class III. Then $n_I = (a_2, \dots, a_{k-i}, a_{k-i+1}, \dots, a_k, b_{i+1}, \dots, b_{i+z-1}) = (a_2, \dots, a_{k-i}, b_1, \dots, b_{i+z-1})$ is an arbitrary node from the route of class I with length $\leq k$ and $n_{III} = (a_m, \dots, a_k, c_2, c_3, b_1, \dots, b_{m-3})$ an arbitrary node from the route of class III with length $k+2$.

Suppose the two routes are not node disjoint. Then two nodes (say the arbitrary chosen nodes) must be equal. So : $n_I = n_{III}$ OR $(a_2, \dots, a_{k-i}, a_{k-i+1}, \dots, a_k, b_{i+1}, \dots, b_{i+z-1}) = (a_2, \dots, a_{k-i}, b_1, \dots, b_{i+z-1}) = (a_m, \dots, a_k, c_2, c_3, b_1, \dots, b_{m-3})$.

Fig. 3 shows two routes from x to y . n_I Denotes the arbitrary node on a route of class I, and n_{III} denotes the arbitrary nodes on a route of length $k+2$ (class III). We assumed that $n_I = n_{III}$. Because the routes are generated with increasing length, we know that the $\text{dist}(n_{III}, y) > \text{dist}(n_I, y)$. Furthermore $\text{dist}(x, n_{III}) > \text{dist}(x, n_I)$.

Fig. 3: Two routes from x to y .

By assuming $n_I = n_{III}$ we induce a repeating pattern in the letters of n_I and n_{III} . In n_I the repetition factor $g = m - z$ so we can define g equivalence classes. In n_{III} the repetition factor $h = i - (m - z) + 2$ with h equivalence classes. If g and h have a common factor then Lemma L1 states that $b_k = c_3 = a_{k-i}$ which contradicts with the in_neighbour inequality. If g and h have no common factor Lemma L2 states that this pattern violates the Kautz inequality. ■ {Proof case I/III}

5. Conclusion.

Kautz graphs form a class of interconnection networks with interesting properties such as: diameter at most $d \log N$ (for $N = d^D + d^{D-1}$), the degree is independent of the network

size, the network is fault-tolerant, it can embed standard computation graphs and has a simple routing algorithm. In this paper we present an algorithm for finding the d node disjoint path between arbitrary nodes in the network. The algorithm is specified in the language Unity. We have proven that the routes found with this algorithm are node disjoint.

A simple hardware implementation of the algorithm follows directly from the Unity specification [11]. The hardware has the structure of a systolic array.

The properties of a Kautz graph makes it a candidate for an interconnection network of large scale parallel computer systems.

References.

- [1] Berge C.: "Graphs and hypergraphs"; North-Holland Publishing Co. 1973.
- [2] Bermond J.C., Homobono N., Peyrat C.: "Large Fault-Tolerant Interconnection Networks", Graphs and Combinatorics, 1989.
- [3] de Bruijn N.G.: "A combinatorial problem"; Koninklijke Nederlandse Academie van Wetenschappen Proc. A49, pp 758-764; 1946.
- [4] Chandy K.M., Misra J.: "Parallel Program Design"; Addison-Wesley Publishing Company; 1988.
- [5] Dally W.J.: "A VLSI Architecture for Concurrent Data Structures", Ph.D. thesis, Computer Science, California Institute of Technology, 1987.
- [6] Fiol M.A., Yebra J.L.A., Alegre I.: "Line digraph iterations and the (d,k) digraph problem"; IEEE Trans. on Computers C-33, pp 400-403; 1984.
- [7] Imase M. and Itoh M.: "A design for directed graphs with minimum diameter.", IEEE Trans. on Comp., vol c-32, pp 782-784, 1983.
- [8] Imase M., Soneoka T., Okada K.: "A fault-tolerant processor interconnection network" (original in Japanese); translated in Systems and Computers in Japan, vol 17, no 8 pp 21-30, 1986.
- [9] Kautz W.H.: "Bounds on directed (d,k) graphs. Theory of cellular logic networks and machines", AFCRL-68-0668 Final report, pp 20-28, 1968.
- [10] Smit G.J.M., Havinga P.J.M., Jansen P.G.: "Generating node disjoint routes in Kautz digraphs", University of Twente, int. rep. INF-90-43, 1990
- [11] Smit G.J.M., Havinga P.J.M., Jansen P.G., de Boer F., Molenkamp E.: "On hardware for generating routes in Kautz digraphs", submitted for publication.