Faculty of Mathematics and Physics
Charles University, Prague

Searching for Edge Disjoint Paths on Hypercube-like Topologies

Doctoral Thesis

Petr Kolman

September 1998
Prague

Let not the wise man glory in his wisdom,
let not the mighty man glory in his might,
let not the rich man glory in his riches;
but let him who glories glory in this, that
he understands and knows me, that I am
the LORD who practices steadfast love,
justice, and righteousness in the earth; for
in these things I delight, says the LORD.

Bible

This research was partialy supported by the Grant Agency of the Czech Republic under grant No. 102/97/1055 and by European Community under project ALTEC-KIT INCO-COP 96-0195.

Acknowledgements

I would like to thank Jiří Sgall for plenty of hours he was willing to spend with me discussing the problems this work deals with. I greatly appreciated his encouragement when some of my proofs and conjectures turned up to be erroneous. I thank my advisor Luděk Kučera for introducing me the delicate de Bruijn graphs, and for focusing my attention to them. I am grateful to my brother David who helped me to improve my English writing. Also, at the very beginning of my undergraduate years he was often the first one whom I asked for a help when I did not understand the new material I was taught at school. Last but not least I am grateful to my parents for their many sided support during the years of my studies and, actually, during the whole of my life.

Supervisor: doc. RNDr. Luděk Kučera, CSc

Referees: RNDr. Ivan Havel, CSc.,
Mathematical Institute of the Academy of Sciences
doc. Ing. Pavel Tvrdík, CSc.,
Czech Technical University, Department of Computer Science

Contents

Chapter 1

Introduction

The problem of on-line searching for disjoint paths on a network comes from several different practical applications. More generally, it is a problem of establishing communication links between desired pairs of nodes on a given network.

A long time ago this problem emerged in the research of telecommunication networks. Consider a telephone switchboard. It has to provide connections for calls that arrive over time, without knowledge about the calls coming in future, and the established calls cannot conflict with each other. The telephone central is represented as a network with two distinguished sets of nodes called inputs and outputs. It is nonblocking if every unused input can be connected by a path through unused edges to any unused output, regardless of which inputs and outputs have been already connected. Sometimes it is quite interesting when two pairs of telephoning people share (unwillingly) the same link, but usually customers of the telephone companies prefer this not to happen. A lot of effort was given to the design of nonblocking networks of sizes as small as possible. Some of the results are surveyed later on. The contemporary telephone centrals have to deal with the same problem, even though the way of transmitting the calls has changed.

Later on, basically the same problem appeared in the research of large scale parallel computers. Information exchange is a fundamental part of any parallel computation. The processing elements are connected via a network (e.g. tree, mesh, hypercube, butterfly) and over time they raise requests for communication with other processing elements. Usually, it is not possible to predict the communication pattern in advance. Instead, communication requests have to be served dynamically according to how they come from the algorithm. Several different techniques were suggested for realizing the data transfer between processors. They can be roughly divided into three categories: Packet routing, circuit switching and wormhole routing. They are not limited to the interconnection networks of parallel computers but some of them are also used in other local and wide area networks.

In packet routing, a message along with its destination address is placed in a packet and the packet moves through the nodes of the network. Each packet is completely received in a node before it is forwarded to the next node. No fixed connection path between the source and destination nodes is established. The transmission time grows at least linearly with the length of the traveled path.

While packet routing is suitable for bounded size messages, it cannot be used efficiently for large volume communications. A more efficient way to transmit such information is through circuit switching. In contrast to packet routing, in circuit switching the entire path from source to destination has to be reserved and only then the data transmission is allowed. At the end of the transmission, the path is released. The data are transmitted in a pipeline fashion and therefore the total transmission time does not depend so much on the distance between source and destination. The other advantage over packet routing is in avoiding the need for large buffers in the intermediate nodes. This method is very common also in the already mentioned telephone switching networks.

The wormhole routing stands in between the packet routing and the circuit switching. A message is routed through the network like a worm, occupying several consecutive links and nodes (depending on its length) on a path from its source to its destination. When the head moves on the entire message moves as well to the free space created. On the other hand, when the head is blocked the tail stops as well. The wormhole routing scheme has to deal with the problem of deadlock when the "worms" are blocking each other in such a way that no one of them can move forward.

In this work we concentrate on the circuit switching which is directly related to the problem of disjoint paths. More information about the packet and wormhole routing can be found in literature, e.g. [50, 26, 20, 22, 23, 44].

In the last years we have been witnessing an enormous growth of the Internet network. Alongside this growth the development of network technologies is going on and the transmission speed of the new high-speed networks is several orders of magnitude higher than that of the existing low-speed networks. This brings new difficulties. The high speed of data transmission leads to a situation when a small delay of a high–rate bit stream exceeds the available buffer space. Also the demands on networks are changing. Recent trends show the growing need for guaranteed quality of service (QoS) rather than just for good average properties (e.g. video on demand, video teleconferencing, real-time database applications). Bandwidth reservation techniques, such as virtual circuit routing, try to answer both of these challenges (e.g. virtual channels are a fundamental part of the ATM technology). The problem of searching for disjoint paths emerges again, although in this case the situation is more general: the paths – virtual channels – are not required to be totally disjoint but several paths may share the same link, according to the bandwidth requirements. The exact correspondence is in a special case when the virtual channels require the entire bandwidth of the links [6, 7]. Again, as in the case of telephone networks or parallel computers, the communication pattern of incoming requests is unpredictable and the requests have to be served on-line.

The QoS requirements bring in another important issue. Since the network resources (bandwidth) are limited and the bandwidth requirements of the established connections have to be fulfilled, some requests have to be rejected. This is a problem of call admission control, i.e. which calls to accept and which ones to reject.

Our aim in this work is to examine some of the common graphs with respect to their nonblocking properties. We do not try to design any new networks. The topologies considered include the Beneš network, a variation of the butterfly, the hypercube and the de Bruijn network. All of these topologies or their close variants were used in practice as interconnection networks for real parallel computers and some of them were also suggested for use in ATM switches (e.g. [47, 51]).

The other problem addressed in this work, as the title suggests, is that of edge disjoint paths. This problem has been shown by Karp [30] to be NP-complete. Different approximation algorithms were given for it. It was also studied in the on-line setting which we will do here. This is closely bound with the call admission problem. The considered topologies are again the the Beneš network, the hypercube and the de Bruijn network. Since there is no hope for good algorithms for general networks (Section 2.3), it is a reasonable setting.

The outline of the work is as follows. In Chapter 2 different topologies and the related problems will be defined in a detail, and important known as well as our results will be surveyed. Each of the following chapters is devoted to a different topology, namely to the Beneš network, to the hypercube and to the de Bruijn network, and to the previously mentioned problems on them. The last chapter gives some open problems. A part of results described in this work was presented at the 6$^{\text{th}}$ European Symposium on Algorithms in August 1998 [35].

Chapter 2

Preliminaries

2.1 Topologies and Embeddings

For a graph G, we denote $V(G)$ the set of vertices and $E(G)$ the set of edges. If there is no danger of confusion we omit the parameter G and use only V and E. There is a few parameters that are often used to compare different topologies (i.e. graphs). The parameters we are interested in include obviously the number of nodes, V, and the number of edges edges, E, then the maximal degree or, in a case of directed graphs, the maximal in and maximal out degrees, and the diameter which is the length of the maximal shortest path between any two vertices. The bisection width is the minimum number of edges that have to be removed in order to disconnect the graph into two halves with identical (within one) numbers of nodes. Many of the properties of different topologies mentioned in this section are discussed in a more detail in the Leighton's textbook [36]. Since the theory of nonblocking networks usually deals with directed graphs whereas the edge disjoint paths problem is mostly considered on undirected graphs, definitions for both cases are given in the following text.

The n-dimensional directed *butterfly* network is defined as follows [36, p.454]. The nodes correspond to pairs (w, i), where i is an integer, $0 \leq i \leq n$, and w is an n-bit binary number. Usually i is called the *level* of the node and w its *row*. There is an edge from (w, i) to (u, j) if and only if $j = i + 1$, and w and u are either identical, or they differ precisely in the j^{th} bit from the left. Given any 0–level vertex $(u, 0)$ and any n–level vertex (v, n), there is exactly one path from $(u, 0)$ to (v, n). In the undirected variant the direction of edges

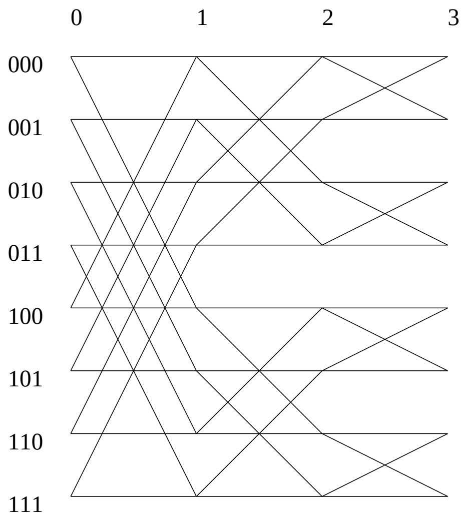

Figure 2.1: Three dimensional butterfly

is simply removed (Fig. 2.1). The n-dimensional directed *Beneš* network B_n consists of two back-to-back butterflies, with all the edges oriented in the same direction ('from left to right'). Overall there are $2n+1$ levels. The number of nodes is $(2n+1)2^n$, the number of edges is $2n2^{n+1}$. The in and out degree of all but the 0 and $2n$ level nodes is two, the 0–level nodes have only two outgoing edges and similarly the $2n$–level nodes have only two incoming edges. Notice that from any vertex only (some) vertices on higher levels are reachable. Most of the time we will use the Beneš network in a special way, namely that we will consider only paths between 0–level and $2n$–level nodes (which is the way how the nonblocking theory uses it). In this case all paths have the same length $2n$. Again, in the undirected variant the direction of edges is simply removed (Fig. 2.2). In this case the maximal degree is four and the diameter is $2n$. The bisection width of N-node Beneš network is $\theta(N/\log N)$.

The Beneš network has a simple recursive structure. One n-dimensional Beneš network B_n contains two $(n-1)$–dimensional Beneš networks B_{n-1} as subgraphs (just remove the edges between levels 0 and 1, and $2n-1$ and $2n$). In the directed case, there is no path from any of these subgraphs to the other. This property will be often used.

The n–dimensional directed *hypercube* H_n consists of $N = 2^n$ vertices, namely of vertices named by all possible binary strings of length n. There

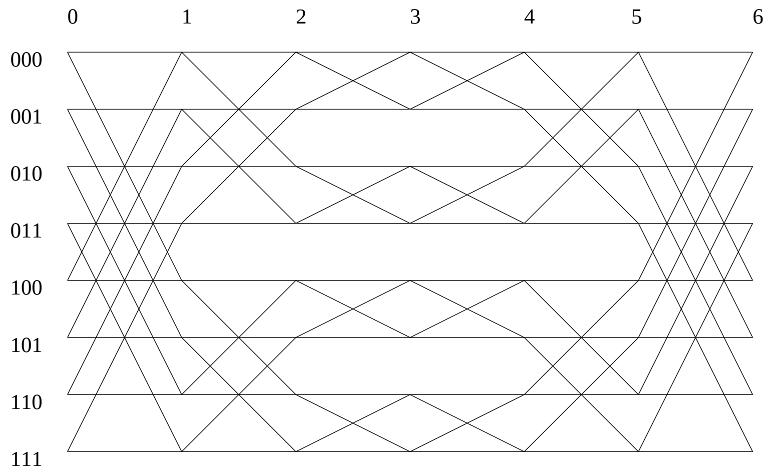

Figure 2.2: Three dimensional Beneš network

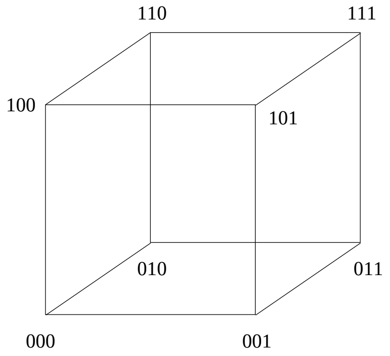

Figure 2.3: Three dimensional hypercube

is an edge from u to v if and only if u and v differ in precisely one bit. The total number of edges is $n2^n$, the in and out degree of all vertices is n and diameter is also n. In the undirected variant, the two opposite edges between the same two vertices are replaced by a single edge (Fig. 2.3), thus the number of edges is $n2^{n-1}$. The bisection width of the undirected hypercube is $N/2$. The disadvantage of the hypercube is the high, non-constant degree.

There is a very close relation between the hypercube and the butterfly (and therefore also with the Beneš network). Take a butterfly, merge all vertices on the same row into a single vertex, omit edges that would become loops and merge multiple (resp. doubled) edges. Then you get exactly the hypercube (Fig. 2.4). This similarity of these two topologies will be used in

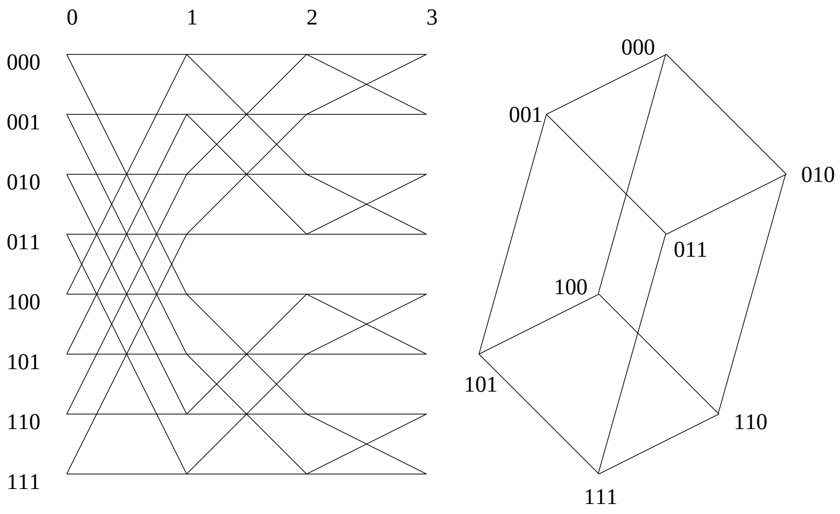

Figure 2.4: Relation between butterfly and hypercube

Chapter 4 to get some results for the hypercube.

The last graph we will define here is the de Bruijn graph [17]. The binary n–dimensional directed *de Bruijn* graph $B(2, n)$ has the same set of vertices as the hypercube, that is all possible strings of n bits. The set of edges is totally different. There is an edge from $u = u_1 \cdots u_n$ to $v = v_1 \cdots v_n$ if and only if $v_i = u_{i+1}$ for $1 \leq i < n$, and v_n is either 0 or 1 (Fig. 2.5). That is, there is an edge from u to v, if either v is a left cyclic shift of u, or v is a left cyclic shift of u with the last bit v_n switched to $1 - u_1$. The number of edges is 2^{n+1}, the in and out degree is 2 in all vertices, i.e. a constant in contrast to n in the hypercube. The diameter is n and the bisection width is $\theta(N/\log N)$. Again, in an undirected variant the direction of edges is simply removed.

Most of these topologies were used in real parallel computers. The butterfly was used as an interconnection network for the BBN Butterfly and a similar Delta network for the NYU Ultracomputer and IBM RP3, the hypercube for Caltech Cosmic Cube and Mark I, II, III, Intel iPSC/1, iPSC/2 and iPSC/860, nCUBE, Thinking Machines CM-1 Connection Machine and CM-2 Connection Machine. Some of them were also suggested for use in ATM switches (e.g. [47, 51]).

In Chapter 4 and 5, when dealing with the hypercube and the de Bruijn graph, we will consider several embeddings of other graphs into them. Let

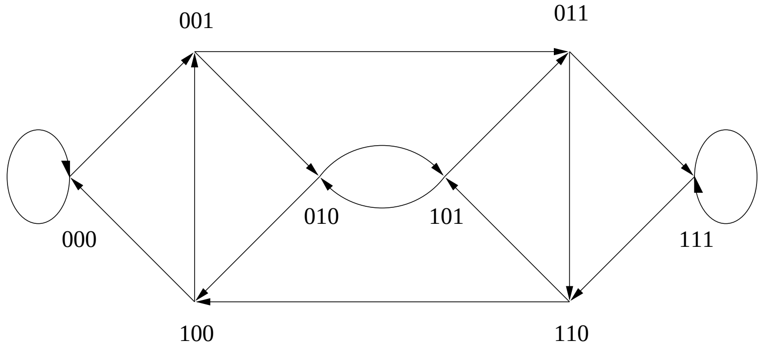

Figure 2.5: Three dimensional de Bruijn graph

G and H be two directed graphs. An *embedding* of the graph G into the graph H is a mapping f of the vertices of G into the vertices of H, together with a mapping g of edges of G into paths in H such that g assigns to each edge $(a, b) \in E(G)$ a path from $f(a)$ to $f(b)$ in H. When dealing with undirected graphs the definition is analogous. From parameters that are used for describing the quality of an embedding we will use the following. The *load* is the maximum number of vertices of G mapped to a single vertex of H. The *congestion* is the maximum over all edges e of H, of the number of edges of G mapped to a path passing through e. The *dilation* is the length of the longest path in H to which an edge of G is mapped. The *expansion* is the ratio between the number of vertices in H and in G. For embeddings of the butterfly and the Beneš networks into the de Bruijn we consider an additional parameter. The *front-end path dilation* is the maximal ratio of length of a path between first and last level nodes in the original graph, and the length of its image in the de Bruijn graph. The maximum is taken over all paths between first and last level nodes in the original graph. This definition is motivated by the fact that often only these paths are considered when dealing with these topologies.

Usually when constructing embeddings it is of interest to have load and expansion equal to 1, and to minimize congestion and dilation. There are known several embeddings of grids and hypercubes into the de Bruijn with the load and expansion 1 [28]. Unfortunately they suffer from a large, non-constant congestion $\log(|V|)$. In our case of dealing with nonblocking properties of graphs, such embeddings cannot be applied to transfer results obtained for one topology to the other, since they would introduce conflicts between the paths. What we need are embeddings with congestion 1, or if we lower slightly our requirements, embeddings with constant congestion. On the other hand, the dilation is of lesser importance for us. Embeddings with constant congestion and constant front-end path dilation of the butterfly and the Beneš network into the de Bruijn are described in Chapter 5, at the cost of increasing the expansion.

2.2 Nonblocking Networks

A *network* is a directed graph with two subsets of nodes called inputs and outputs. Its *depth* is the length of the maximal shortest path between inputs and outputs and its *size* is the number of edges in it. The *order* of a network is the number of its inputs and outputs. A N*–network* is a network with N inputs and N outputs, i.e. a network of order N. The inputs of the butterfly and Beneš networks are the 0–level nodes, the outputs the last level nodes (i.e. the n–level nodes in the butterfly and $2n$–level nodes in the Beneš network). A network is supposed to provide connections between the inputs and outputs. There are two types of *requests* for a network: (i) to make a connection between a given input and a given output, and (ii) to destroy a specified connection. If not stated otherwise, connection requests will be considered only between unused inputs and outputs (i.e. each input and output may appear in at most one connection request).

Most of the time we will consider blocking of edges. Therefore the following definitions of nonblocking networks are given only for this case. However, when needed (once in this section and in Sections 3.2 and 3.5) analogous definitions will be assumed also for a case of blocking nodes.

A N–network is called *strict sense edge nonblocking* [45] if, no matter what edge disjoint paths are established in it, it is possible to connect every unused input to any unused output through unused edges. If there is an algorithm that is able to serve every sequence of requests in an edge disjoint fashion, starting from an empty network, the network is called *wide sense edge nonblocking* [45]. If there is an algorithm that is able to serve each sequence of connection requests only, we call the network *weak sense edge nonblocking*. A network is *edge rearrangeable* [15, 45] if for any set of requests given at once it is possible to serve all the requests in an edge disjoint fashion. Clearly, any strict sense nonblocking network is wide sense nonblocking, wide sense nonblocking is weak sense nonblocking, weak sense nonblocking is rearrangeable.

The definition of strictly nonblocking network is not very reasonable for networks where it is possible to block all or a huge fraction of edges by a single path. Such networks surely will not be strictly nonblocking of large order. This definition was originally introduced for networks where all paths between given input and output are of the same length. This is not the case of the hypercube and the de Bruijn graph. In fact, if all paths between any pair of input and output in a network are of the same length then there are no cycles in the the network. But there are plenty of cycles in the hypercube and the de Bruijn. Even if the use of cycles in paths would be prohibited it is still possible to block by a single path many edges redundantly. Neither other possible attempt to modify the definition, namely to allow use of shortest paths only, is not appropriate (e.g. consider a hypercube and any path for request from $00\cdots0$ to $11\cdots1$, and then requests between every two consecutive nodes on this path). Therefore we will not deal with *strictly* nonblocking properties of the hypercube and the de Bruijn graph.

What is the minimal size of a nonblocking network? The lower bound $\Omega(N \log N)$ for the number of edges in a node rearrangeable N–network was proved by Shannon [46]. We will recall it shortly and then show how to modify it for the edge disjoint case. Consider a rearrangeable N–network with m edges. There are $N! \geq (N/e)^N$ distinct permutations on N elements, each corresponding to a different set of requests for the network. For each of the permutations there must be a different set of paths between the inputs and outputs. The total number of different sets of node disjoint paths in a network with m edges, is bounded from above by the number of subsets of the m edges, which is 2^m. Thus, if the network is rearrangeable, it is necessary that

$$2^m \geq N! \quad ,$$

which implies the desired $m \geq N \log_2 N - N \log_2 e$. When we are concerned with edge disjoint paths, the total number of different sets of such paths in a network with maximal degree d can be bounded from above by $(d+1)^m$. The reason is that each edge is either not included in any path, or if it is, then there are d possibilities for its successor in the path. This gives a lower bound of $\Omega(N \log N / \log d)$ for the number of edges, which is $\Omega(N \log N)$ for constant degree networks.

The corresponding upper bound for rearrangeable networks was proved by Beneš [15] and Beizer [11]. They constructed bounded degree rearrangeable network of this size. Moreover, there is a very simple recursive algorithm for establishing the paths on the Beneš network. It was given by Waksman [52]. It works even if each input and output is used in two requests. Consider n-dimensional Beneš network and an arbitrary set of such 2^{n+1} requests. The algorithm specifies for each request whether to use the upper or lower $(n-1)$–dimensional Beneš subnetwork. By doing this, the problem is divided into two problems, half size each. It works as follows. Suppose that for no or even number of requests it is already specified whether to use the upper or lower B_{n-1}. Let r be the lastly considered request and let p be the other request from the same input. If p is not yet served, then the algorithm decides about it. In fact, there is no choice for it since p has to use the other B_{n-1} than the request r does. As the next request, take the one with same output as p. Again, there is no choice for it. If p is already served, or if no requests are yet served, then choose the next request arbitrarily from those not yet served, and then continue similarly.

We have to show that the algorithm is always able to decide about all the requests without encountering a conflict on any edge. Notice that after deciding about every second request the number of them assigned to the upper and lower B_{n-1} is the same. Also, at this moment each output is either not yet used or it is already used by both its requests. Thus no conflicts can appear for odd requests. Similarly, after deciding about every odd request each but one input is either not yet used or it is already used by both its requests. Thus the only possible place for conflict to appear is for an even request on a first level edge if the input of it coincides with the single input used by only one request at this moment. But such a conflict would be a contradiction with the even distribution of requests to the upper and lower B_{n-1}. Notice that at the end all edges of the network are used.

It is worth mentioning at this point that the butterfly network is not rearrangeable at all. In fact, there is a permutation for which the congestion of all paths between inputs and outputs is $\sqrt{N/2}$, for $N = 2^n$ (e.g. [36, p.515]). For simplicity assume n is odd and consider the bit reversal permutation, i.e. we want to connect $(x_1 \cdots x_n, 0)$ to $(x_n \cdots x_1, n)$. Since the paths between inputs and outputs are uniquely given in the oriented butterfly, all paths from inputs of the form $x_1 \cdots x_{\frac{n-1}{2}} 0 \cdots 0$ have to pass the butterfly edge $\left((0 \cdots 0, \frac{n-1}{2}), (0 \cdots 0, \frac{n+1}{2})\right)$. The number of such inputs is $2^{\frac{n-1}{2}} = \sqrt{N/2}$.

A lot of attention was given to the problem of rearrangeability of the directed hypercube. Szymanski [49] conjectured that the H_n is rearrangeable for every n. He proved it for $n \leq 3$. His conjecture was even stronger, namely that H_n is rearrangeable even if each connection is required to take the shortest path. A counterexample to the conjecture about rearrangeability via shortest paths was given by Lubiw [40] (in a 5–dimensional hypercube; there is a counterexample even in a 4–dimensional hypercube [24]). On the other hand there are also several positive results. It follows from the rearrangeability of the Beneš network that a directed hypercube with doubled edges is rearrangeable (e.g. [40]). Choi and Somani [19] proved that doubling edges only in one fixed dimension is sufficient for the hypercube to be rearrangeable. Fertin [24] showed that the 4–dimensional hypercube is rearrangeable when non-shortest paths are allowed. Other result, related to optical routing, was proved by Gu and Tamaki [27]: Any permutation can be partitioned into 2 partial permutations so that each of them can be routed via edge disjoint paths. For undirected hypercube there is a similar result due to Aumann and Rabani [2]: Any permutation can be partitioned into 16 partial permutations so that each of them can be routed via edge disjoint paths.

An naturally arising question about the Beneš network is whether it is also strict sense nonblocking? It is not, as we shall see later. So the next question is whether it is at least possible to choose a subset of the 0–level nodes and a subset of the $2n$–level nodes such that the network is strictly nonblocking with respect to only these inputs and outputs, and, what is the maximal size of such a subset. We show that the size cannot be larger than a fraction $3/4$ of all 0–level nodes. On the other hand if in n–dimensional Beneš network only a suitable chosen fraction $1/n$ of inputs and outputs is used (e.g. the first $2^n/n$ inputs and outputs), then it is strict sense nonblocking. The corollary of it is that we have a very simple strict sense nonblocking network with $N = 2^n$ inputs and outputs, namely a $(n+\log n+1)$–dimensional Beneš network. It has $O(N \log^2 N)$ edges and depth $O(\log N)$. An almost identical network, the Cantor network (cf. [18]), was shown, by other method, to be strictly nonblocking by Melen and Turner [43].

There are assymptoticaly smaller strictly nonblocking networks. Bassalygo and Pinsker [9, 10] proved the existence of an assymptoticaly optimal network with $O(N \log N)$ edges. However, their result relies heavily on expander graphs and is not constructive. Only explicit construction of expanders [41, 42] yielded the network. Also other constructions of nonblocking networks and their generalizations are based on expander graphs [1, 21]. If we compare the constants hidden in the big O notation in the result of Bassalygo and Pinsker [10] and in our result we find that for less than about 2^{15} inputs and outputs the Beneš network is the smaller one. Thus, from practical point of view the Beneš network is at least comparable.

As stated earlier, our primary aim is not to design new nonblocking networks, but to study nonblocking properties of some well known and used networks. The first set of problems can be defined like this: given a graph G, which will be either the Beneš network, the hypercube, or the de Bruijn graph in our case, give k as large as possible such that there are two disjoint subsets V_1 and V_2 of nodes $V(G)$ each of size k, such that the graph G is rearrangeable (or nonblocking in strict, wide or weak sense) k–network. Preferably give the subsets V_1 and V_2 as well. We have already seen that for the case of rearrangeability and the Beneš network the question is totally answered. For the stronger nonblocking properties we give the subsets V_1 and V_2 of size $2^n/n$ as mentioned earlier. This result is also modified for the hypercube. For the n–dimensional de Bruijn graph it is proved that for any constant $c > 0$ it is not k-rearrangeable network, for $k \geq (\log_2 3 + c)2^{n+1}/n$. Since the de Bruijn graph shows to be quite a difficult graph to deal with in a case of establishing many paths, the requirements on the nonblocking network will be lowered slightly in this case. Instead of insisting on edge disjointness of the paths we will allow constant congestion. Then we show that the de Bruijn graph is $(2^n/n^{1+c})$-rearrangeable network and $(2^n/n^{2+c})$–widely nonblocking network, where c is an arbitrary constant $c > 0$. Thus the bounds on the rearrangeability almost match each other.

2.3 Disjoint Paths Problem and On-line Algorithms

The problem of establishing disjoint paths that the theory of nonblocking networks deals with is closely related to the *problem of edge disjoint paths*. It is defined as follows. Given an undirected graph $G = (V, E)$ and a set T of k pairs of nodes s_i, t_i, $1 \leq i \leq k$, decide whether there exist k edge disjoint paths $P_1, \cdots, P_k$, such that the path P_i connects s_i and t_i. It has been shown by Karp [30] that this is an NP–complete problem. There is an optimization version of it, called *maximum disjoint paths problem*, which is simply to find the maximum subset of T for which there exist the edge disjoint paths. Different approximation algorithms were proposed for it.

We will consider the maximum disjoint paths problem in an on-line setting. In this case the requests for connections (s_i, t_i) arrive one after another, and for each of them the algorithm has to decide whether to reject it or to accept it. In the later case it has also to provide a path for it that is disjoint with all the paths established previously. In the sequence of requests the same nodes may appear several times. What we are interested in is the competitive ratio of the algorithm. The *competitive ratio* of a deterministic on-line algorithm, whose effort is to get as maximal gain as possible, is defined to be

$$c = \sup_{\sigma} \frac{OPT(\sigma)}{ON(\sigma)} \quad ,$$

where the supremum is taken over all possible sequences of requests, and $ON(\sigma)$ is the number of requests accepted by the on-line algorithm and $OPT(\sigma)$ is the number of requests accepted by an optimal off-line algorithm [16]. For a randomized on-line algorithm it is defined similarly, just the number $ON(\sigma)$ of accepted requests is replaced by the expected number $E\left[ON(\sigma)\right]$ of accepted requests, that is

$$c = \sup_{\sigma} \frac{OPT(\sigma)}{E\left[ON(\sigma)\right]} \quad .$$

We analyze our algorithms against oblivious adversary. That is the adversary cannot take into account the already realized random decisions of the algorithm, when presenting a next request of the sequence [48, 16], cf. [13, 12].

Our second problem can then be stated like this: For a given graph G, which is either the Beneš network, the hypercube, or the de Bruijn, propose an on-line algorithm for the maximum disjoint paths problem on G, with competitive ratio as small as possible.

This problem was already considered for several different topologies. Garay, Gopal, Kutten, Mansour and Yung [25, 29] dealt with it on an N–node line network and they presented a randomized $O(\log N)$-competitive algorithm for it. A matching lower bound was given by Awerbuch, Bartal, Fiat and Rosén [6]. They considered this problem also on a tree network and they designed a randomized algorithm for an N–node tree with a competitive ratio $O(\log N)$. In an unpublished version of their paper [5] they also suggested an $O(\log d)$–competitive algorithm for any diameter d tree. Another such algorithm was described by Awerbuch, Gawlick, Leighton and Rabani [7], and they also proved the matching lower bound. In the same paper they proposed a randomized $O(\log N \log\log N)$–competitive algorithm for a two dimensional $N \times N$ mesh and proved a lower bound $\Omega(\log N)$ on the competitive ratio of a randomized algorithm on the two-dimensional mesh. The optimal algorithm for the mesh was described by Kleinberg and Tardos [34], and it was generalized for a special class of planar graphs ("densely embedded, nearly-Eulerian graphs"). Awerbuch et al [7] dealt also with the hypercube and they gave an $\Omega(\log\log N)$ randomized lower bound for it.

Most of these algorithms suffer from a drawback that only the *expected* competitive ratio is good. It may happen that they compute a very poor solution with a very high probability, and a good solution with a very low probability. Leonardi, Marchetti-Spaccamela, Presciutti and Rosén [38] deal with this problem a they propose other randomized algorithms for trees and meshes with optimal competitive ratios that obtain a good solution with a high probability.

All the algorithms mentioned up to this point are randomized. This reflects the fact that the lower bounds for deterministic algorithms for these topologies are much higher. For the line network there is a trivial lower bound $\Omega(N)$ (e.g. [4]), which can be easily generalized to $\Omega(d)$ for any diameter d tree. Awerbuch et al [7] mention a deterministic $\Omega(\sqrt{N})$ lower bound for $N \times N$ mesh, by Blum, Fiat, Karloff and Rabani. Kleinberg [31] provides an alternative proof. The known deterministic on-line algorithms for the maximum disjoint paths problem, with polylogarithmic competitive ratios, are for expander graphs and for the hex. Kleinberg and Rubinfeld [32] described an $O(\log N \log\log N)$–competitive algorithm for any bounded degree graph with strong expansion properties and Awerbuch et al [7] an $O(\log^2 N)$–competitive algorithm for the hex.

There was also an effort to design algorithms that work well on any network topology. The deterministic lower bound $\Omega(N)$ for the line shows that there is no hope for deterministic algorithms with reasonable competitive ratio. Bartal, Fiat and Leonardi [8] prove this effort to be vain even for randomized algorithms by giving an $\Omega(N^\epsilon)$ lower bound for randomized on-line algorithms on general networks, where $\epsilon = 2/3^{1-\log_4 3}$.

The problem appears to be much easier when requests may allocate only a small fraction of link capacities. If each request may require at most a fraction $O(1/\log N)$ of link capacities, then there is $O(\log N)$–competitive algorithm for general topologies, by Awerbuch, Azar and Plotkin [4]. The authors also gave a matching lower bound for this setting.

More detailed overview about most of the briefly mentioned results can be found in the recently published textbook by Borodin and El-Yaniv [16, Ch.13], or in the survey article by Leonardi [37].

As stated earlier, we concentrated on the case when requests require the full capacity of the links, that is on the classical maximum disjoint paths problem. Awerbuch, Azar, Fiat, Leonardi and Rosén [3] proposed a general technique that enables to extend results for this problem also for optical routings with multiple wavelengths. We describe $O(\log N)$–competitive randomized and deterministic algorithms for the Beneš network for the special case of requests between the first and last level nodes only, and $O(\log^2 N)$–competitive barely random algorithm for the general case. The algorithms are then modified to randomized and deterministic $O(\log^2 N)$–competitive algorithms for the N-node hypercube and arbitrary requests. The best results obtained for the de Bruijn graph include a deterministic $O(\log^{3+c} N)$–competitive algorithm, for arbitrary $c > 0$.

Throughout this work we consider only on-line algorithms for *permanent* call problem. However, the algorithms can be extended for both known and unknown limited duration calls. The idea is to use the "classify and randomly select technique" by Lipton and Tomkins [39], cf. [6]. The resulting algorithms are randomized and the competitive ratios are greater by an additive factor of $\log T$ and $\log T \log\log^{1+\epsilon} T$, resp. where T is a known and unknown bound on the ratio between the minimum and maximum duration calls, resp. [16, p. 315].

2.4 General Technique

We will conclude this chapter with two lemmas describing general technique that will be used several times for obtaining on-line algorithms for maximum disjoint paths problem. Both of them are rather simple but they will appear useful. The first one transfers results for the off-line variant of the the problem to the on-line variant.

Lemma 1 *If there exists an α–approximating solution for the maximum disjoint paths problem on a graph $G = (V, E)$ that uses only paths of length at most d (where d may depend on $|V|$ and $|E|$), then there is an αd-competitive on-line algorithm.*

Proof: Consider a greedy on-line algorithm ALG: given a request for a path between a and b, accept it if and only if there is a free path between

a and b of length at most d. For the accepted request use any of the found limited length paths.

The only possible reason that ALG rejects a request between a and b that is part of the α–approximating solution, is that each path between a and b intersects in an edge with some previously established path. Since each request accepted by ALG may cause later on rejection of at most d requests belonging to the α–approximating solution, we conclude that ALG is αd–competitive. □

The other lemma transfers results from "maximum *almost* disjoint paths problem" (i.e. if a constant congestion is allowed instead of insisting on edge disjointness) to maximum disjoint paths problem.

Lemma 2 *If there is a c–competitive on-line algorithm ALG for the problem of maximum paths with congestion at most k on a graph $G = (V, E)$ and the accepted paths are of length at most d, then there is a ckd–competitive on-line algorithm for the maximum disjoint paths problem.*

Proof: The idea is the same as in the previous proof. We just slightly modify the algorithm ALG. All requests rejected by ALG will be rejected by the new algorithm as well. Consider a request between a and b accepted by ALG. If the path designed for it by ALG does not intersect with paths of the previously accepted requests by the new algorithm, the new algorithm accepts it. In this way each request accepted by the new algorithm may cause rejection of at most kd requests accepted by ALG. □

Chapter 3

Beneš Network

This chapter forms, in some sense, a foundation of the presented work. Most of the results for both the hypercube and the de Bruijn graphs are derived from the properties of the Beneš network.

At the beginning of this chapter the notion of i–similarity is introduced, and necessary and sufficient condition for existence of a free path in a partialy blocked Beneš network is given. This yields an efficient algorithm for searching for a path in a partialy blocked network, which is described in Section 3.1. The partially nonblocking property of the Beneš network is given in Section 3.2 and its consequences for the on–line routing in Section 3.3. Section 3.4 states some stronger results about Beneš networks of very small dimensions. Finally, in Section 3.5 it is shown that the Beneš network is not so good in the stronger nonblocking properties as it is in the rearrangeability.

Let's start with the definition of similarity. Numbers a and b are said to be *i–similar* if their binary representations agree on i least significant bits. Their *similarity* is 2^{i-1}. Two requests or paths from a to b and from c to d are *i–similar* if either a and c, or b and d are i–similar. Note that in the definition we do not care about where the paths actually go.

Lemma 3 *Suppose there are k paths established in the Beneš network B_n. Then these paths are edge disjoint if and only if there is no Beneš subnetwork B_i, $0 \leq i \leq n$, and no paths p_1, p_2 such that both p_1 and p_2 are passing through the B_i, and they are $(i+1)$–similar.*

Proof: $\Rightarrow$ By contradiction. Suppose the paths are edge disjoint and there is B_i with two paths p_1 and p_2 passing through it, with their *outputs*, without loss of generality, $(i+1)$–similar. Then, after $n+i$ steps they have to reach a node on a line that is i–similar with their outputs and, since in each B_i there is only one such line, the paths reach the same node after $n+i$ steps. But because the outputs are $(i+1)$–similar both paths have to continue via the same edge from the B_i, which is a contradiction.

$\Leftarrow$ We will show that if the paths are not edge disjoint then there is B_i and two paths passing through it with their inputs or outputs $(i+1)$–similar. Let s and t be the two paths that use the same edge e. Suppose that the edge is closer to the outputs (without loss of generality; otherwise we would for a while replace the inputs with outputs and the other way round). Let B_i be the Beneš subnetwork that the edge e is incident to and not part of it (i.e. e is going out of B_i). Then, s and t are $(i+1)$ similar and they are passing through the same B_i. □

Lemma 4 *Suppose there are k paths established in the Beneš network B_n and there is a request for the next path from a to b. Then a path for this request cannot go in an edge disjoint fashion with the previously established paths through a vertex u on the medium level n if and only if there is an already established path p such that*

- *p is $(i+1)$–similar with the request $a \to b$, and*
- *p is passing through a medium level vertex v such that $\lfloor u/2^i \rfloor = \lfloor v/2^i \rfloor$.*

Proof: $\Rightarrow$ Let us establish the path from a to b through u. Then the path will use some edge already used by some path p. Let i be a maximal number such that p and $a \to b$ are $(i+1)$–similar. Then both of them pass through the same B_i (Lemma 3), and thus $\lfloor \frac{u}{2^i} \rfloor = \lfloor \frac{v}{2^i} \rfloor$, i.e. we have the desired path.

$\Leftarrow$ The path p disables all medium level nodes in B_i that it passes through from being a medium level nodes for path from a to b. Since $\lfloor \frac{u}{2^i} \rfloor = \lfloor \frac{v}{2^i} \rfloor$, we know u is one of these vertices. □

The lemma says that each path p, that is $(i+1)$–similar with the given request $a \to b$, disables the interval $I_p = \langle k2^i, \ldots, (k+1)2^i - 1 \rangle$ of 2^i medium level nodes from being a medium level node of the path from a to b, where $k = \lfloor \frac{v}{2^i} \rfloor$ and v is the medium level node of p (Fig. 3.1). Given description of the path p by its input in_p, its output out_p and its medium level vertex $medium_p$, it is possible to compute the interval $I_p = \langle k2^i, \ldots, (k+1)2^i - 1 \rangle$ in constant time. We have the following theorem:

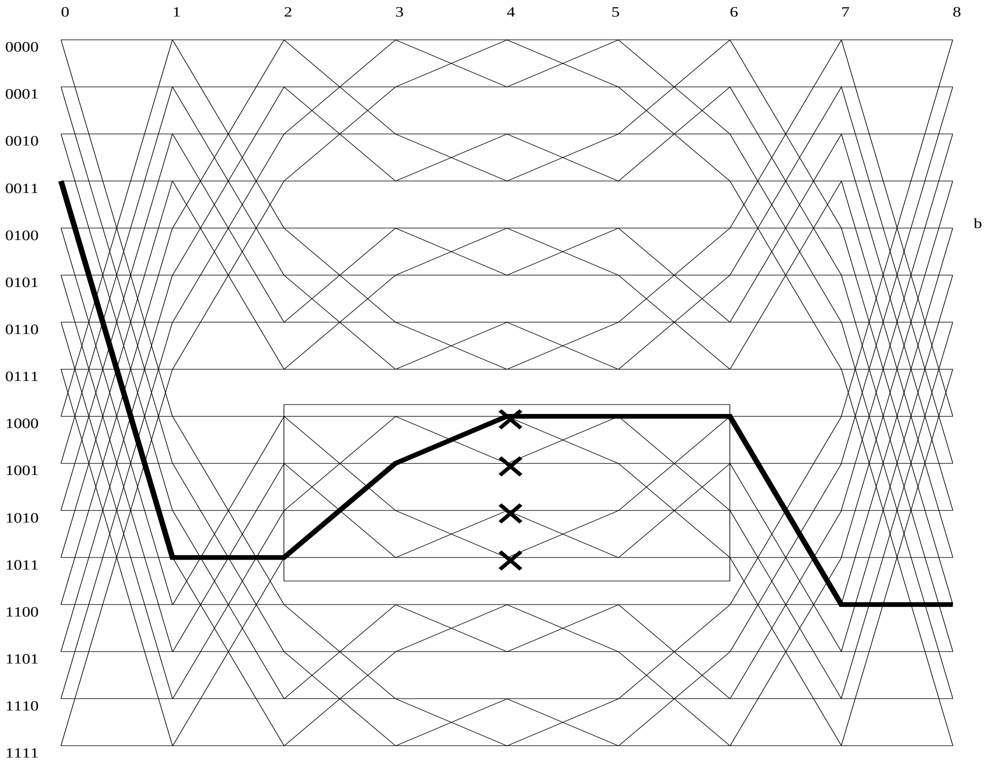

Figure 3.1: The path p from input 3 to output 12 disables medium level nodes $8, 9, 10, 11$ from being a medium level node for a path from a ($0 \leq a \leq 15, a \neq 3$) to $b = 4$. The path p and the request $a \to b$ are at least 3–similar, their similarity is $2^{3-1} = 4$, $k = 8/4 = 2$, $I_p = \langle 8, 9, 10, 11\rangle$.

Theorem 5 (Necessary and sufficient condition for existence of a free path) *Given a set P of paths on the Beneš network such that each input and output is used at most once, and a request for a path from input a to output b, there is a free (i.e. along free edges) path from a to b if and only if*

$$\bigcup_{p \in P} I_p \neq \langle 0, 2^n - 1\rangle \quad .$$

Proof: The theorem follows from Lemma 4 and from the previous discussion. $\Box$

Let us note that for two intervals I_p and I_q, corresponding to paths p and q, such that $I_p \cap I_q \neq \emptyset$ and $|I_p| \leq |I_q|$, it is always the case that $I_p \subseteq I_q$.

3.1 Searching for a Path

Theorem 6 *Given a set P of k paths in the Beneš network and a request for a path from input a to output b, it is possible to decide in time $O(k)$ whether there exists a free path for this request, and if it exists to describe it.*

Proof: Due to Theorem 5 it suffices to decide whether the union of intervals I_p covers all nodes on the medium level or not. We shall use an array c of size 2^n to solve the problem. For simplicity we shall assume that this array is initialized to 0, later on we shall see how to avoid this requirement (we would like to avoid it since the initialization itself would take $\Omega(2^n)$ time). By taking any of the uncovered nodes in the medium level, if there is any, as the medium level node for the path from a to b, we have also unique description of the new path.

Algorithm

```
for each path p ∈ P do
    i = max{similarity(a, in_p), similarity(b, out_p)}
    if i > 0 then
        k = ⌊medium_p/2^i⌋
        b_p = k2^i                /* beginning of the interval I_p */
        e_p = (k + 1)2^i − 1      /* end of the interval I_p */
        c[b_p] = max{e_p, c[b_p]}
    endif
enddo
pointer = 0
while (pointer < 2^n) and (c[pointer] ≥ pointer) do
    pointer = c[pointer] + 1
enddo
if pointer < 2^n then "There is a free path via medium level node pointer."
    else "There is no free path from a to b."
endif
```

The values in the array c have the following properties at the end of each round of the first cycle:

- if $c[j] = 0$ then j is not a lower endpoint of interval I_p for any path p considered so far,

- if $c[j] \neq 0$ then $c[j]$ is the upper endpoint of the largest interval with lower endpoint in j.

It follows from this and from the fact that one of two overlapping intervals is always contained in the other that at the end of the algorithm *pointer* points to the minimal index unblocked medium level node, if there is any, or *pointer* $= 2^n$ if there is none. So the algorithm solves the problem.

It remains to show how to avoid the initialization. We will add some information to the array c: whenever we set the value of $c[j]$ we add information about due to which path we set $c[j]$ in this way. Whenever reading $c[j]$ we check whether the information read is really valid. The case that $c[j]$ is not valid corresponds to $c[j] = 0$. This change causes only a constant slowdown.

Since both cycles run at most k times and each round takes $O(1)$ time, (in our model we count for an operation on two n–bits numbers a unit cost) the algorithm runs in time linear in the number of paths. □

For establishing and searching for paths it is also possible to use a parallel breadth first search with running time $O(n)$.

3.2 Nonblocking Properties

Theorem 7 *The Beneš network B_n is a strict sense edge nonblocking network of order $2^n/n$.*

Proof: Let us take as inputs only the 0–level nodes that agree on the $\lceil \log n \rceil - 1$ most significant bits (the leftmost), and similarly as outputs only the $2n$–level nodes that agree on the $\lceil \log n \rceil - 1$ most significant bits. The number of such inputs (and outputs) is $2^{n-(\lceil \log n \rceil - 1)} \geq 2^n/n$. It does not matter what is the actual setting of the $\lceil \log n \rceil - 1$ bits, specifically the setting of the inputs need not be the same as of the outputs.

Suppose there are some paths established between the chosen inputs and outputs. Consider any request $a \to b$ from an unused input a to an unused output b. Our aim is to prove an existence of a free path for it. The proof will rely on Theorem 5. We will show that the sum of similarities of a with all other input nodes is strictly less than 2^{n-1}, and the same for b and output nodes: The number of i–similar and not $(i+1)$–similar inputs is $2^{n-i-\lceil \log n \rceil}$, for $1 \leq i \leq n - \lceil \log n \rceil$. The similarity of each such input is 2^{i-1} and the sum of similarities of all i–similar inputs is $2^{n-\lceil \log n \rceil - 1}$. Since

$(n - \lceil \log n \rceil)2^{n-\lceil \log n \rceil -1} < 2^{n-1}$ and the same holds also for the outputs, we conclude that the sum of similarities of the current request with all already established paths is strictly less then 2^n. So the union $\bigcup_{p \in P} I_p$ of intervals is definitely a proper subset of $\langle 0, 2^n - 1 \rangle$ and there is a free path from a to b. □

The theorem says that we have a strict sense nonblocking network, namely the Beneš network when properly seen, with N inputs and outputs, that has $O(N \log^2 N)$ edges and depth $O(\log N)$, which is not constructed out of expanders.

Consider the following modification of the $(n+\lceil \log n \rceil)$–dimensional Beneš network: replace each path of length $\lceil \log n \rceil$ starting in the first 2^n inputs by a single edge, and remove all other edges in the first $\lceil \log n \rceil$ levels. Do the same for outputs. Then each of the 2^n inputs and outputs is connected to $\lceil \log n \rceil$ copies of n–dimensional Beneš network. This is a Cantor network and it was also proved by other method to be strictly nonblocking by Melen and Turner [43].

In Chapter 4 when dealing with the hypercube, a node disjoint variant of Theorem 7 will be needed as a tool. Lemmas 3 and 4, and subsequently Theorems 5 and 7 can be easily modified for the node disjoint case (basically the difference is that each path blocks twice as much medium level nodes in the node disjoint variant than in the edge disjoint). Here we state only the final theorem that will be used in Section 4.1.

Theorem 8 *The Beneš network B_n is a strict sense node nonblocking network of order $2^{n-1}/n$.*

3.3 Call Admission and Circuit Routing

In this section a slightly different kind of requests will be considered. There will be no destruction requests but only requests for connection. Each request will contain information about its starting and final points, which have to be 0–level and $2n$–level nodes, and about its duration. If the duration of all calls is infinite we speak about permanent call problem, otherwise about known limited duration call problem. There is no restriction on the number of requests originating and terminating in a node. The property of the Beneš network described in Theorem 7 leads to a randomized $O(n)$-competitive on-line algorithm for the n-dimensional Beneš network and to a similar algorithm for the hypercube. We analyze our algorithms against oblivious adversary. It is worth noting than even only weak sense nonblocking property will yield the same results.

Theorem 9 *There is a randomized* $4n$*–competitive algorithm for permanent call problem on the* n*-dimensional Beneš network.*

Proof: The immediate consequence of Theorem 7 is that there is a deterministic 4–competitive algorithm for establishing paths between only the first 2^n inputs and outputs on an $(n + \lceil \log n \rceil)$–dimensional Beneš network: accept a request $a \rightarrow b$ if there is no other accepted request originating in a or terminating in b. Since the outdegree of input nodes and indegree of output nodes is 2 there may be at most 4 requests accepted by optimal algorithm but rejected by our algorithm due to acceptance of the request $a \rightarrow b$.

Consider any such algorithm ALG. Choose uniformly at random an integer r, $0 \leq r < n$. Accept only those requests that are routed by ALG through any of medium level nodes $r2^n, r2^n + 1, \cdots, (r+1)2^n - 1$ and route them on B_n in the same way as they are routed by ALG through the $2n$ medium levels of the higher dimensional network. The expected number of calls accepted by the described algorithm is at least $1/(4n)$ of the number of calls accepted by the optimal off–line algorithm, and all of them are routed via edge disjoint paths. □

With help of the general technique of Lemma 1 it is possible to obtain even a deterministic algorithm with the same (resp. slightly better) competitive ratio.

Theorem 10 *There is a deterministic* $2n$*–competitive algorithm for permanent call problem on the* n*–dimensional Beneš network.*

Proof: It follows from the rearrangeability of the Beneš network that the optimal algorithm uses paths of length at most $2n$. Thus, according to the Lemma 1 the greedy algorithm is $2n$–competitive. □

The algorithm may be extended for requests between arbitrary nodes at the expense of higher competitive ratio, on the undirected Beneš network. The idea is to directly reject all requests between rows, that already have a path terminating on them, and to present remaining requests as requests between first and last level nodes to the algorithm of Theorem 10. If there would be doubled edges in the graph there would be no difficulty with augmenting the paths designed by the algorithm of Theorem 10 to the real source and terminal nodes. Without doubled edges one has to be more careful. The following algorithm is from the class of barely random algorithms, it uses only $4 = O(1)$ random bits.

Let R_i be the set of rows $\{i2^{n-2}, \cdots, (i+1)2^{n-2} - 1\}$, for $i = 0, 1, 2, 3$. At the beginning the algorithm chooses uniformly independently at random two integers $i, j \in \{0, 1, 2, 3\}$. These are the only random decisions of the algorithm. Let h be any of the not chosen integers, say the minimal (but it does not matter) and let $h_1 h_2$ be the binary representation of h. Let ALG be the deterministic algorithm of Theorem 10 for $(n-2)$–dimensional Beneš network B_{n-2}. The decision about acceptance or rejection of a request $(a, k) - (b, l)$ is as follows. If $a \notin R_i$ or $b \notin R_j$, or if $a \in R_i$ and $b \in R_j$ but there is an already accepted request either from the row a or to the row b, then reject. This will be called the first level of decision. If a request is not rejected on this level then a request $a_3 a_4 \cdots a_n \to b_3 b_4 \cdots b_n$ is presented to the algorithm ALG, where $a_1 a_2 \cdots a_n$ is binary representation of a and $b_1 b_2 \cdots b_n$ of b. If ALG rejects, reject $(a, k) - (b, l)$ as well, otherwise accept. Which path to use for an accepted request? It will consist of three parts. The main part is the middle one, designed by ALG for B_{n-2}. It goes from $(\hat{a}, 2) = (h_1 h_2 a_3 \cdots a_n, 2)$ to $(\hat{b}, 2n-2) = (h_1 h_2 b_3 \cdots b_n, 2n-2)$. This middle part is augmented by a short initial part from (a, k) to $(\hat{a}, 2)$ and a short final part from $(\hat{b}, 2n-2)$ to (b, l). The initial part goes as follows:

- if $i, h < 2$ or if $i, h \geq 2$ then it starts along the row a from level k to level 1, from $(a, 1)$ it proceeds along the straight edge to the $(h_1 h_2 a_3 \cdots a_n, 2)$ (notice that $a_1 = h_1$ and $\bar{a}_2 = h_2$ in this case, where $\bar{x} = 1 - x$),

- otherwise it starts along the row a from level k to level 0 (i.e. to $(a, 0)$), and then it continues through vertex $(h_1 a_2 \cdots a_n, 1)$ to the $(h_1 h_2 a_3 \cdots a_n, 2)$i, Fig. 3.2 (notice that $\bar{a}_1 = h_1$).

The final part is designed in the same way on the other side of the Beneš network. Since at most one request from or to a row is accepted, no two initial nor final parts may intersect. Also, the rows of the $(n-2)$–dimensional Beneš subnetwork, that is used for the middle parts of the accepted requests, are disjoint with the initial and final rows R_i and R_j. Thus, all accepted requests are routed via mutually edge disjoint paths.

Theorem 11 *The competitive ratio of the described algorithm is $O(n^2)$ on the n–dimensional Beneš network.*

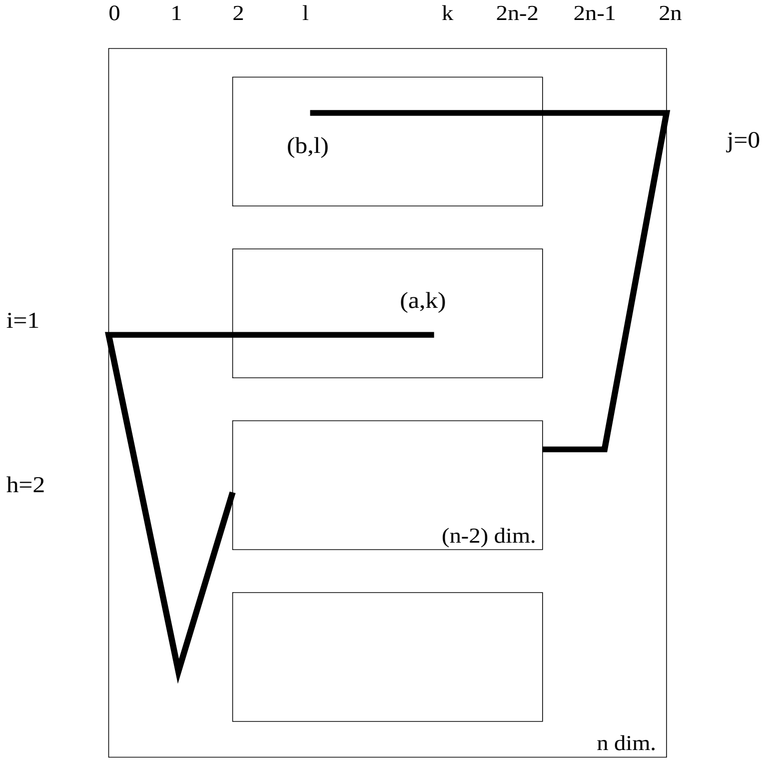

Figure 3.2: The initial and final parts of a path for request between nodes (a, k) and (b, l). The initial part goes from (a, k) to $(a, 0)$ and then through $(h_1 a_2 \cdots a_n, 1)$ to $(h_1 h_2 a_3 \cdots a_n, 2)$. The final parts starts in $(h_1 h_2 b_3 \cdots b_n, 2n-2)$ and through $(h_1 b_2 b_3 \cdots b_n, 2n-1)$ proceeds to $(b_1 b_2 b_3 \cdots b_n, 2n)$ and then finaly to (b, l).

Proof: The expected number of requests between rows R_i and R_j is $1/16$ of all requests and thus the restriction to these requests only does not affect the size of the competitive ratio (which is an expectation) by more than a constant. In the following let us consider only requests from R_i to R_j. For each request accepted by our algorithm there may be at most $O(n)$ other requests accepted by the optimal algorithm but rejected by our algorithm on the first decision level. The ALG used for the second decision level is $O(n)$–competitive, and moreover, if each input and output appears only once in the input sequence, the number of accepted requests is at least a fraction of $1/(2n)$ of all requests. This is exactly the kind of sequence provided to *ALG* in our case. Thus, in total, the competitive ratio is $O(n^2)$. □

3.4 Small Dimension Beneš Networks

In this section some new notation is used. Let $B_1(p)$ denote the Beneš subnetwork B_1 that p is passing through, and $\neg B_1(p)$ the other B_1 belonging to the same B_2. We say that two paths p, q such that $B_1(p) = B_1(q)$ are *parallel* if they do not have a common vertex in $B_1(p)$. When it is allowed for two requests to start in the same node then we speak about $2-1$ requests. Moreover, when two requests may terminate in the same node they are called $2-2$ requests.

Theorem 12 *The Beneš network B_2 is a wide sense edge nonblocking network for $2-1$ requests.*

Proof: Our algorithm makes routing decisions according to five rules:

1. Whenever routing a second request through B_1, do it in a non-parallel way.
2. If there is no path in B_2 then route the request through the upper B_1.
3. If there is only one path p in B_2 then route the request through $B_1(p)$ if possible.
4. If there are two paths p, q then route the request through $B_1(p)$ if possible.
5. Suppose there are three paths p, q, s and it is possible to route the request both through the upper B_1 and the lower B_1. Without loss of generality we may assume that $B_1(p) = B_1(q)$. If by routing the request through $B_1(p)$ no new parallel paths would emerge then route the request through the $B_1(p)$, otherwise through $\neg B_1(p)$.

The rules 2–5 describe decisions about the first step, rule 1 about the second.

Consider a configuration of paths in B_2. If there exist paths c_1, c_2, c_3 such that c_1, c_2 are parallel, and c_3 could pass in a non-parallel way through $B_1(c_1) = B_1(c_2)$ but does not, we call it a *dead configuration.* Two lemmas will be proved that show together the correctness of the algorithm.

Lemma 13 *If the already established paths are not in a dead configuration, then the algorithm is able to route the next request.*

Lemma 14 *The algorithm never establishes a dead configuration.*

Proof (Lemma 13): There is always enough space for routing a request if there are at most two established paths. Therefore assume there are three paths p, q, s in B_2 not in a dead configuration, and a request $a \to b$ is given. If the input a was not used up to this time, then there is a free path from a to b through either the upper or the lower B_1 and the algorithm finds it. The situation is a bit more difficult if a is already used by a path, say by p, since then the request cannot go through $B_1(p)$. If at most one of q, s is passing through $\neg B_1(p)$, or if both q, s are passing through $\neg B_1(p)$ and they are not parallel, then there is a free path from a to b through $\neg B_1(p)$ and the algorithm finds it. The only remaining case is that q and s are parallel and $B_1(p) \neq B_1(q) = B_1(s)$. Assume for a while that q and s are the only paths in B_2 and consider an unused input u. Then u may be connected with exactly one of the two unused outputs by a path passing through $B_1(q)$. Now consider the path p again. From the observation above and from the fact that p, q, s are not in a dead configuration it follows there is free path for the request through $\neg B_1(p)$ and the algorithm finds it. $\neg B_1(p)$ and the algorithm finds it. $\square$

Proof (Lemma 14): From the description of the algorithm it follows that the property is always satisfied whenever a third path is established. Also deleting any path will not create a dead configuration. Suppose now there are three paths not in a dead configuration and a request $a \to b$ is given. There are two ways how the "not dead" property may be broken: by establishing a path c_3, or by establishing c_1 or c_2 (i.e. one of the two parallel paths). By rules 1 and 5 we get that no parallel path will emerge if there is a choice for the first edge of a path from a to b, or if there is no choice for it but there is at most one other path in the B_1 that the request is forced to go through. Neither it happens in these cases that a path c_3 emerges. The only left case is that the request is forced to pass through B_1 with two already established paths in it. Let p be the path that forces the request to do so, that is let p be the path that starts in a. The assumption that the configuration was not dead and the fact that the path p and the request have to use different B_1 yield that neither in this case a dead configuration is created. $\square$

Theorem 15 *The Beneš network B_3 is a wide sense edge nonblocking network for $1-1$ requests.*

Proof: Consider this algorithm. If the endpoint of the request is in the lower half of B_3 then use for the path as the first edge the edge to the lower B_2, otherwise to the upper. By doing it we split the problem into two $2-1$ routing problems on B_2 which we will deal with according to the Theorem 12. $\square$

Corollary 16 *There is a deterministic* 4*–competitive algorithm for permanent routing problem on a* 4*–dimensional Beneš network* B_3*.*

Theorem 17 *The Beneš network* B_3 *is a weak sense edge nonblocking network for* $1-1$ *requests.*

As we noticed at the beginning this is an immediate consequence of the previous theorem. We state it separately because of a very simple direct proof of it. Moreover, our conjecture is that the algorithm described in this proof works for the Beneš network B_4 as well.

Proof: Consider a greedy algorithm: route the given request through as low medium node as possible. It is also possible to look at this algorithm as a depth first search: if the succeeding lower node is reachable continue to it, if it is not but the upper neighbor is reachable, go there, if neither of them is reachable, return to the nearest node from which the upper neighbor was not tested yet.

We will show that this algorithm is always able to establish all 8 paths. Assume there are some paths established and we are trying to establish the next one. There are always two reachable nodes in the first step. These two nodes are connected to four nodes in the next (i.e. second) level. Since we are dealing with B_3, at most one of these four nodes may be blocked by an already established path with a 2–similar input, and at most one of them may be blocked by an already established path with a 2–similar output. So it suffices to show that the algorithm is able to do the third step as well. Suppose it is not able to do it. Let u be one of the reachable nodes in the second level. Then it is a primarily dead node and one of the outgoing edges from u is blocked by an already established path p (i.e. in_p is 1–similar with a), and the other is going to a node with a path q going through it, such that out_q is 1–similar with b. Either p is going to the upper neighbor and q to the lower, or the other way round. But none of these possibilities may happen, since there would be three 1–similar paths in the same B_1, and thus two of them would be 2–similar, which is a contradiction. $\square$

3.5 Negative Results

The results in this section show that the Beneš network is not so good in the stronger nonblocking properties as it is in rearrangeability.

Theorem 18 *The Beneš network B_n is not a wide sense edge nonblocking network of order 2^n for $2-2$ requests, for $n \geq 2$.*

Proof: We have to show that there is no algorithm for establishing the paths when the requests are presented on-line. Suppose, by contradiction, there is such an algorithm. We will describe a sequence of requests such that it is unable to establish the edge disjoint paths for them. Let a, c be two different inputs and b, d, e three different outputs. Consider the following sequence of requests:

1. $a \to b$, let suppose, without loss of generality, that it passes through the lower B_{n-1},
2. $a \to e$, it has to pass through the upper B_{n-1},
3. $c \to d$, if it passes through the lower B_{n-1}, then there is no place left for request $c \to e$; if it passes through the upper B_{n-1}, then there is no place left for $c \to b$.

It is easy to see that on B_1 this argument cannot be used and that B_1 is a wide sense edge nonblocking network of order 2 for $2-2$ requests. □

Theorem 19 *The Beneš network B_n is not a wide sense node nonblocking network of order 2^n for $1-1$ requests, for $n \geq 3$.*

Proof: Again a sequence of requests will be provided that no deterministic algorithm is able to serve. Consider the following sequence of requests, for $N = 2^n$ (Fig. 3.3):

1. $a \to b$, $a \geq 5N/8$, $b \geq 3N/4$, let suppose, without loss of generality, that it passes through the lower B_{n-1},
2. $c \to d$, $d = b - N/4$, if it would go through the upper B_{n-1} then there is no place left for $x \to y$, $x = a - N/2$, $y = d - N/2$, so let us suppose it passes through the lower B_{n-1},

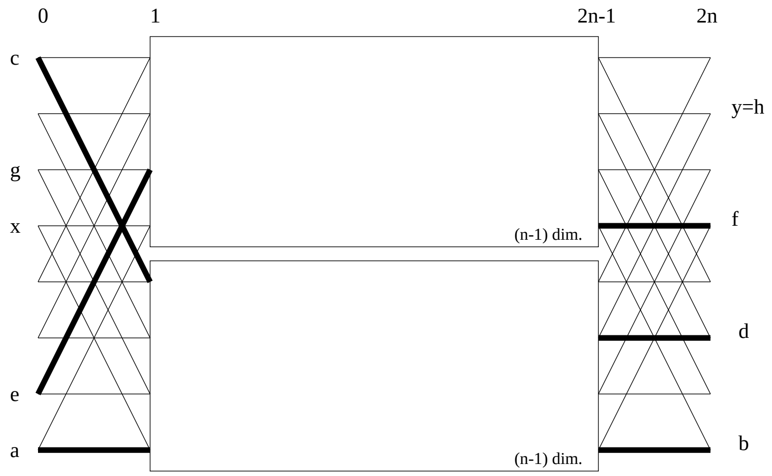

Figure 3.3: The Beneš network B_n is *not* a wide sense node nonblocking network of order 2^n for $1-1$ requests. Situation after establishing the first three paths from the proof of Theorem 19. There is no place left for path from g to h.

3. $e \to f$, $e = a - N/8$, $f = b - N/2$, it has to go through the upper B_{n-1}, since the node in the level $2n-1$ of the lower B_{n-1}, from which the output f is reachable, is already blocked,

4. $g \to h$, $g = a - 5N/8 = e - N/2$, $h = d - N/2$ – there is no place left for it.

□

Theorem 20 *The Beneš network B_n is not a strict sense edge nonblocking network of order 2^n for $1-1$ requests, for $n \geq 3$.*

Proof: We will provide a description of 6 paths and a nonsatisfyable request for connection (Fig. 3.4). For a full description of a path it suffices to state the input, the output and n bits $b_1, \cdots b_n$, where the i-th bit says whether the i-th edge on the path leads to the upper ($b_i = 0$) or lower ($b_i = 1$) subbutterfly B_{n-i}. In fact for our purposes it will suffice to provide only the first three bits b_1, b_2, b_3. Moreover, all inputs and outputs used will be an integer multiple of $N/8$, so in the description we will write just the multiple of $N/8$:

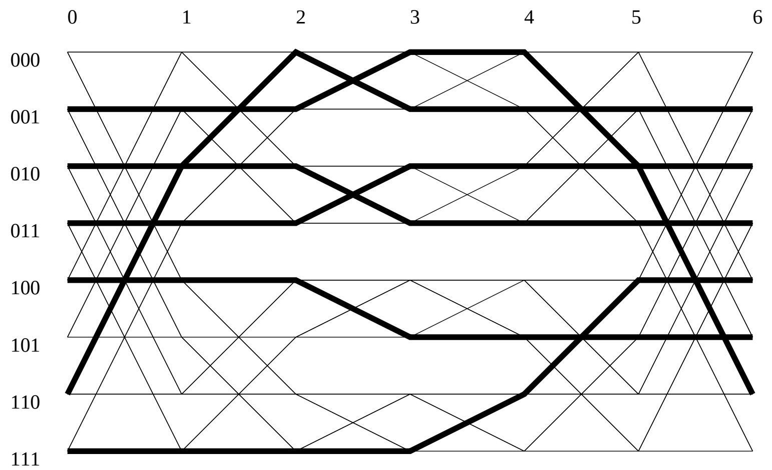

Figure 3.4: The Beneš network B_n is *not* a strict sense edge nonblocking network of order 2^n for $1-1$ requests. The six described paths disable establishing a path from 0 to 0.

input	output	$b_1b_2b_3$
1	2	000
2	3	011
3	6	010
4	5	101
6	1	001
7	4	111

Then it is impossible to establish an edge disjoint path for request $0 \to 0$. □

Corollary 21 *The Beneš network B_n, $n \geq 3$, is not a strict sense edge nonblocking network of order s, for any $s > \frac{3}{4}\,2^n$.*

Proof: If we allow more then 6 out of any 8 $(n-3)$–similar 0–level nodes to be input nodes and more then 6 out of any 8 $(n-3)$–similar $2n$–level nodes to be output nodes, then the construction from Theorem 20 (or a similar one) shows the network is not a strict sense nonblocking. □

Chapter 4

Hypercube

Most of the results described in this chapter are derived from the similarities of the hypercube and the butterfly described in Section 2.1. The first section of this chapter states a bound on widely nonblocking properties of the hypercube and the second section describes algorithms for call admission and path selection.

4.1 Nonblocking Properties

In Section 2.2 some known results about the rearrangeability of the hypercube were mentioned. Here we concentrate on its nonblocking properties.

Theorem 22 *The n–dimensional directed hypercube is widely edge nonblocking network of order $2^{n-2}/n$.*

Proof: Embed an $(n-1)$–dimensional butterfly into the upper half of the hypercube (i.e. into nodes whose binary representation begins with 1) and other $(n-1)$–dimensional butterfly into the lower half. Use the embedding described in Section 2.1 (resp. a similar one for the directed case). Consider these two butterflies as an $(n-1)$–dimensional Beneš network and apply the result of Theorem 8. The *strict* sense nonblocking property of the Beneš network is weaken to the *wide* sense nonblocking property of the hypercube, since on the hypercube we cannot use *any* paths but we have to follow the paths on the embedded Beneš network.

The reason for relying on Theorem 8 instead of Theorem 7 is that in the described embedding of the Beneš network into the hypercube the medium level Beneš nodes have two copies. In each direction these copies are connected by only one edge. Since *two* paths may pass through a medium level Beneš node when dealing with *edge* disjoint paths we would encounter difficulties on the hypercube. By referring to the *node* disjoint variant of the same theorem, the problem disappears. □

4.2 Call Admission and Circuit Routing

Theorem 23 *There is a randomized $O(n^2)$-competitive algorithm for permanent call problem on the n-dimensional directed hypercube.*

Proof: Let A be the subset of the hypercube nodes whose binary description begins with 1, and let B be the other subset. Let f be a function on n–bit strings that returns all but the first (leftmost) bit, i.e. $f(x_1x_2\cdots x_n) = x_2\cdots x_n$. All the requests are divided into four categories according to whether they start and end in A, in B, or start in A and end in B, or start in B and end in A. At the beginning the algorithm uniformly and randomly chooses one of the four categories, say category C. All requests not belonging to C are automatically rejected. The expected number of requests in C is 1/4 of all requests.

Consider a request from a to b from category C. If there is some other accepted request with a or b as a source or sink, the request $a \to b$ is rejected. Otherwise our algorithm simulates the behavior of the algorithm ALG from Theorem 10 (or 9, resp.) on an $(n-1)$–dimensional Beneš network: If ALG accepts the request from $f(a)$ to $f(b)$, our algorithm tries to accept $a \to b$ as well. It has to provide a path for it. Assume that the accepted request belongs to category "from A to B". As we noticed earlier the n–dimensional hypercube contains two $(n-1)$–dimensional hypercubes as subgraphs. Let H^1 be the hypercube on vertices A, and H^2 the hypercube on vertices B. As in the proof of the previous theorem we embed the first $n-1$ levels of the simulated $(n-1)$-dimensional Beneš network, i.e. $(n-1)$–dimensional butterfly, into H^1, and the last $n-1$ levels of the $(n-1)$–dimensional Beneš network, i.e. the other butterfly, into H^2 (the medium level Beneš nodes are mapped both to H^1 and H^2). The path for the request is as follows. It first follows the first $n-1$ edges of the path from $f(a)$ to $f(b)$ on the $(n-1)$–dimensional butterfly embedded into H^1, then moves from one copy of the medium level Beneš node to the other one, and finally follows the last $n-1$

edges of the path on the Beneš network on H^2. The only possible place for conflict with path of a previously accepted request is on the edge connecting the two copies of the medium level Beneš node. If such conflict appears, the request $a \to b$ is rejected. When dealing with request from category "from A to A" we prolong the path by a traverse edge from H^2 to H^1 at the end. Requests from the other two categories are dealt with similarly.

At least a fraction of $1/(4 \cdot 2n)$ of requests accepted by an optimal algorithm for the hypercube proceeds to the second level of our algorithm, i.e. to the algorithm ALG for the Beneš network. The algorithm ALG is $O(n)$ competitive on the Beneš network, and our algorithm accepts at least half of requests accepted by ALG. Moreover, if each input and output appears only once in the input sequence, the number (expected number, resp., if algorithm of Theorem 9 is used) of accepted requests is at least a fraction $1/(2n)$ ($1/n$, resp.) of all requests. This is exactly the kind of sequence provided to ALG in our case. Thus, in total, the number of requests accepted by the algorithm for the hypercube is at least a fraction $\Omega(1/n^2)$ of all requests accepted by the optimal algorithm, and our algorithm is thus $O(n^2)$–competitive.

The same result holds also for the undirected hypercube with doubled edges. The proof is analogous. □

Other way how to obtain an $O(n^2)$–competitive algorithm is, similarly as in the case of the Beneš network, to use the general technique of Lemma 1. The advantage of it over the previous algorithm is that it is deterministic and that it works even for undirected hypercube (without doubled edges). It makes use of the properties of the hypercube proved by Aumann and Rabani [2] (a slightly stronger result for directed hypercube was proved by Gu and Tamaki [27]):

Theorem 24 (Gu and Tamaki) *Any permutation on the undirected hypercube can be splited into* 16 *parts such that each of them can be realized using edge disjoint paths (of length at most* $2n$*).*

What is important for us (not stated directly in the original theorem), is that all paths that are used to route the permutation are of length at most $2n$ (basically the paths are obtained by simulating paths on the Beneš network, the difficult part is how to split the permutation in the parts).

Theorem 25 *There is a deterministic* $O(n^2)$*–competitive algorithm for permanent call problem on the* n*–dimensional undirected hypercube.*

Proof: Consider a request $a - b$. If there is an already accepted request with terminal node a or b then reject the request $a - b$ (first decision level). Otherwise, if there is a free path between a and b of length at most $2n$, accept the request and use the founded path for it (second decision level).

If each hypercube node would appear in at most one request in the sequence then by Lemma 1 the algorithm would be $32n$–competitive. This is not the case and any node may appear repeatedly in the requests. The first decision level ensures that requests that proceed to the second decision level already form such a sequence. For each request accepted by our algorithm there may be at most $2n$ requests rejected by it on the first decision level that the optimal algorithm may accept. In total, the algorithm is $O(n^2)$–competitive. $\square$

Chapter 5

De Bruijn Graph

Though much more complicated, the strategy for dealing with the de Bruijn graph resembles the way we dealt with the hypercube. It happens easily on the de Bruijn graph that a shortest path between two nodes is blocked. Which path to use for a request if the shortest one is not at disposal? The structure of the de Bruijn graph is not very perspicuous at the first sight. In this work the following strategy is used for dealing with problem. We embed a Beneš network (of smaller dimension) into the de Bruijn graph, and for all requests we use paths that follow paths on the embedded Beneš network. As mentioned already in Chapter 2, an embedding of constant (preferably equal to one) congestion is needed to ensure that the designed paths are (almost) edge disjoint.

The outline of the chapter is as follows. In Section 5.1 limits of the rearrangeability of the de Bruijn graph are proved. Section 5.2 describes the promised embedding of the Beneš network which yields also results about the widely nonblocking property of the graph. Section 5.3 consists of technical lemmas that are needed for the algorithm for call admission and path selection in Section 5.4.

5.1 Bound on the Rearrangeability

Let us concentrate on nonblocking properties of the de Bruijn graph and first on its rearrangeability. What is the maximal k such that the n–dimensional de Bruijn graph is a k–rearrangeable network (i.e. it is possible to find two disjoint subsets of nodes I and O of size k such that given any permutation specifying connections between these two subsets it is possible to establish them in an edge disjoint fashion)? The next theorem bounds the size of k in directed de Bruijn graph.

Theorem 26 *For $k = (1+c)2^{n+1}/n$, where c is an arbitrary constant $c > 0$, the directed n–dimensional de Bruijn graph is not rearrangeable k–network.*

A simple lemma will help us to prove it:

Lemma 27 *Given any two subsets A and B of nodes of size at least 2^d in the directed n–dimensional de Bruijn graph, there exist $a \in A$ and $b \in B$ such that the shortest path from a to b is of length at least d.*

Proof (Lemma 27): Since the outdegree of nodes is 2, given any de Bruijn node, at most $2^d - 1$ nodes are accessible from it in less than d steps. Thus, there is a node $a \in A$ from which not all nodes from B are accessible. Let b be one of them. □

Proof (Lemma 26): Suppose, by contradiction, that the graph is k–rearrangeable, for $k = (1 + c)2^{n+1}/n$ and some $c > 0$. Let I and O be the two subsets. With the help of the previous lemma a set of requests will be described that the network is not able to satisfy. To get the first request (a_1, b_1) from this set we apply the Lemma 27 to the sets I and O. Suppose that we have already defined l requests from our set. To get the next request apply Lemma 27 to sets $A_l = I - \bigcup_{i=1}^{l}\{a_i\}$ and $B_l = O - \bigcup_{i=1}^{l}\{b_i\}$. In this way $(1 + c/2)2^{n+1}/n$ requests are defined. Each of them requires a path of length $n - O(\log n) > n/(1 + c/2)$ for n sufficiently large. In total these requests require more edges than the de Bruijn graph contains, which is a contradiction. □

There is also a slightly weaker variant of the theorem for the undirected de Bruijn graph.

Theorem 28 *For $k = (\log_2 3 + c)2^{n+1}/n$, where c is an arbitrary constant $c > 0$, the undirected n–dimensional de Bruijn graph is not rearrangeable k–network.*

Proof: The proof is again by contradiction. Suppose the graph is k–rearrangeable, for $k = (\log 3 + c)2^{n+1}/n$ and some $c > 0$. The difference is that the number of nodes that are accessible in a given number of steps is slightly higher in the undirected graph:

Lemma 29 *Given any two subsets A and B of nodes of size at least $2 \cdot 3^{d-1}+2$ of undirected de Bruijn graph, there exist $a \in A$ and $b \in B$ such that the shortest path from a to b is of length at least d.*

Proof: Starting from any node, the number of nodes reachable in exactly b steps is at most $4 \cdot 3^{b-1}$. The total number of nodes reachable in at most $(d-1)$ steps is $2 \cdot 3^{d-1}+1$. □

In the same way as in Theorem 26 we define $(\log 3 + c/2)2^{n+1}/n$ requests. Each of them requires a path of length at least $n/\log 3 - O(\log n) > n/(\log 3 + c/2)$. Again, in total this is more than $n2^n$, which is a contradiction. □

5.2 Embedding of a Butterfly into de Bruijn Graph and Nonblocking Properties

Let us introduce some notation first. A move along any edge in the de Bruijn graph corresponds to an operation on the binary representation of the starting node. There are two basic operations:

- $L(x), x \in \{0,1\}$ - left shift, forget the most important bit (the leftmost) and add x as the least important bit (the rightmost),
- $R(x), x \in \{0,1\}$ - right shift, forget the least important bit and add x as the most important bit.

When the added bit is the same as the forgotten, the operation is a plain cyclic shift and the argument will be occasionally omitted.

For a description of a path on de Bruijn graphs, a sequence $X_1(x_1)X_2(x_2) \cdots X_j(x_j)$, $X_i \in \{L,R\}, x_i \in \{0,1\}$ of these two operations will be used. To give an example consider node $u = 101011$ of 6-dimensional de Bruijn graph. Then sequence $L(0)L(1)R(1)$ corresponds to path $101011 \rightarrow 010110 \rightarrow 101101 \rightarrow 110110$.

For brevity subsequences of the form $L(x_1)L(x_2)\cdots L(x_j)$ and $R(x_1)R(x_2) \cdots R(x_j)$ will be often shorted to $L_j(x_1x_2\cdots x_j)$ and $R_j(x_j \cdots x_2x_1)$. Thus the path from the previous example will be described as $L_2(01)R(1)$, given the starting node u.

For any two (directed or undirected) de Bruijn nodes $\mathbf{u}$ and $\mathbf{v}$, let

$$d_L(\mathbf{u},\mathbf{v}) = \min_j \{\text{there is a path } L_j(x_1x_2\cdots x_n) \text{ of length } j \text{ from } \mathbf{u} \text{ to } \mathbf{v}\} \ ,$$

and similarly let

$$d_R(\mathbf{u},\mathbf{v}) = \min_j \{\text{there is a path } R_j(x_1x_2\cdots x_n) \text{ of length } j \text{ from } \mathbf{u} \text{ to } \mathbf{v}\} \ .$$

In a directed de Bruijn graph $d_L(\mathbf{u},\mathbf{v})$ is the length of the shortest path from $\mathbf{u}$ to $\mathbf{v}$, and similarly $d_R(\mathbf{u},\mathbf{v})$ is the length of the shortest path from $\mathbf{v}$ to $\mathbf{u}$. This is generally not the case in an undirected de Bruijn graph. We have always $d_R(\mathbf{w},\mathbf{v}) = d_L(\mathbf{v},\mathbf{w})$.

Theorem 30 *It is possible to embed an $(n-2\lceil\sqrt{n}\ \rceil)$-dimensional butterfly into an n-dimensional de Bruijn graph with load 1, congestion 2 and dilation 2.*

Proof: Consider at first a simple mapping of an n-dimensional butterfly into an n-dimensional de Bruijn graph. It does not have the desired properties but it will give an insight in the latter mapping. A node $(u_1u_2\cdots u_n, i)$, $1 \le i \le n$, is mapped to a de Bruijn node $(u_{i+1}u_{i+2}\cdots u_nu_1u_2\cdots u_i)$, a node $(u_1u_2\cdots u_n, 0)$ to the same node as $(u_1u_2\cdots u_n, n)$. The nice property about this mapping is that it has dilation only 1. The bad thing is that its load is $\Omega(\log n)$. The problem is that most of information about levels of the original nodes is lost by the mapping and therefore many of them are mapped to the same place. The idea how to solve this difficulty is to insert a special sequence C between bits u_n and u_1 in the mapping. The sequence C, or rather its position, should supply the information about level of the original butterfly node. To guarantee load 1 it is necessary to ensure that in each de Bruijn node, to which some butterfly node is mapped, there is only one such subsequence C.

Let $k = n - 2\lceil\sqrt{n}\rceil$ denote the dimension of the butterfly we want to embed and let $l = \lceil\sqrt{n}\rceil - 1$. Let C be a sequence of $\lceil\sqrt{n}\rceil$ zeros. Consider a node $u = (u_1u_2\cdots u_n, i)$ and let $c = \lfloor i/l \rfloor$. Then u is mapped to de Bruijn node

$$\begin{aligned}(u_{i+1}\cdots u_{(c+1)l}\ 1\ u_{(c+1)l+1}\cdots u_k\ 1\ C\ 1\ u_1u_2\cdots u_l\ 1\ u_{l+1}u_{l+2}\cdots u_{2l}\clubsuit \\ \clubsuit 1\ \cdots\ 1u_{(c-1)l+1}u_{(c-1)l+2}\cdots u_{cl}\ 1\ u_{cl+1}\cdots u_i)\end{aligned}$$

i.e. after every l^{th} bit 1 is inserted, after the last bit u_k the sequence C enclosed in two 1's is appended, and then $i+c$ left shifts $L_{i+c}(\,)$ are performed. The sign $\clubsuit$ is not part of the mapping, it only denotes how to concatenate the two lines. Concerning the edges, they are mapped to any of the shortest paths between corresponding nodes. We shall see that the shortest path is uniquely given.

The fact that the load is one follows from the structure of the de Bruijn nodes to which the butterfly nodes are mapped. Consider a de Bruijn node $v = (v_1 v_2 \cdots v_n)$. If there is no $1\ C\ 1$ subsequence in $v_1 v_2 \cdots v_n$, or there are at least two such subsequences, then no butterfly node is mapped to v. Otherwise it is uniquely given what butterfly node is mapped to v.

By this mapping two neighboring butterfly nodes are mapped either to two neighboring de Bruijn nodes, or to two de Bruijn nodes at distance two, therefore the dilation is 2 as desired. In the case that the original edge is mapped to a path of length 2 the intermediate node is uniquely given, and it is not used as an image of any other butterfly node. From this observation and the fact that the load is 1 and the dilation 2 we get that congestion is at most 2. □

Corollary 31 *The undirected n–dimensional de Bruijn graph is rearrangeable $(2^{n-2\lceil\sqrt{n}\,\rceil})$-network, and widely nonblocking $(2^{n-2\lceil\sqrt{n}\,\rceil-\lceil\log n\rceil})$-network, if congestion 4 is allowed.*

Proof: According to the Theorem 30 embed two $(n-2\lceil\sqrt{n}\,\rceil)$-dimensional butterflies (i.e. an $(n-2\lceil\sqrt{n}\,\rceil)$-dimensional Beneš network) into the n-dimensional de Bruijn graph and apply the results for the Beneš network. The *strict* sense nonblocking property of the Beneš network is weaken to the *wide* sense nonblocking property of the de Bruijn, since on the de Bruijn graph we cannot use *any* paths but we have to follow the paths on the embedded Beneš network. □

When dealing with butterfly (or Beneš) networks, the 0–level nodes are often considered as inputs and the last–level nodes as outputs. The disadvantage of the previously described embedding is that the dimension of the embedded butterfly is too low compared to the de Bruijn dimension. This drawback is partially removed in the next embedding at the cost of increasing dilation. However, when dealing with paths from inputs to outputs only, the dilation or rather the so–called front–end path dilation remains constant.

Theorem 32 *It is possible to embed an $(n-3\lceil\log n\rceil-6)$-dimensional butterfly into an n-dimensional de Bruijn graph with load 1, congestion $O(1)$, dilation $O(\log n)$ and front-end path dilation $O(1)$.*

Proof: For simplicity we shall assume that n is a power of 2. Let us denote $k = n - 3\log n - 6$, the dimension of the butterfly we want to embed. We divide the nodes of the butterfly into $\lceil k/\log n \rceil$ disjoint groups. The division is done according to the levels of the nodes: a node (u, i) belongs to group $\lfloor i/\log n \rfloor$. Let A be a sequence of three bits 010 and B a sequence of 101. Consider a butterfly node $(u_1u_2\cdots u_k, i)$. Let us denote $g = \lfloor \frac{i}{\log n} \rfloor$ the group number of the node and $r = i - g\log n$ the *relative level number*, which is the number of the node's level inside its group. Let $w_1w_2\cdots w_{\log n}$ be the binary representation of g (padded with 0's from the left side) and let $m = g\log n = i - r$. Then we map the node $(u_1u_2\cdots u_k, i)$ to a node

$$\underbrace{u_{m+r+1}u_{m+r+1}\cdots u_{m+\log n}u_{m+\log n}}_{doubled}|B|\overbrace{w_1w_2\cdots w_{\log n}}^{group}|\overbrace{u_{m+\log n+1}\cdots u_{m+2\log n}}^{memory}\clubsuit$$

$$\clubsuit u_{m+\log n+1}\cdots u_{k-1}u_k|u_1u_2\cdots u_m|A|\underbrace{u_{m+1}u_{m+1}\cdots u_{m+r}u_{m+r}}_{doubled}.$$

The sign $|$ only marks the several subsequences, it is not part of the mapping. To put it verbally, to get the image of $(u_1u_2\cdots u_k, i)$ we start with the first m bits $u_1u_2\cdots u_m$, insert the sequence A after them, double each of the following $\log n$ bits $u_{m+1}\cdots u_{m+\log n}$, insert sequence B followed by the $\log n$ bits $w_1w_2\cdots w_{\log n}$ of the group number, then we continue with the sequence $u_{m+\log n+1}\cdots u_{m+2\log n}$ and end by the last $k - (m + \log n)$ bits $u_{m+\log n+1}\cdots u_{k-1}u_k$. We finish by performing $m + |A| + 2r$ left shifts $L_{m+|A|+2r}(\ )$. Since the sequence of $\log n$ bits $u_{m+\log n+1}\cdots u_{m+2\log n}$ appears two times, even when overwriting one of them the information about these $\log n$ bits is not lost. Therefore we call the first of these two subsequences a memory. We will denote this mapping of nodes by $f(\ )$.

How to map the edges? Consider an edge between two nodes u and v from the same group. These two nodes are mapped to two nodes of the de Bruijn graph that are at distance two from the other. The edge (u, v) is mapped to the unique path of length two between $f(u)$ and $f(v)$. This path is called *short embedded edge*.

The difficulty is with edges between nodes from different groups. The danger of using too short paths for them is that the congestion would not be constant. Here, paths of length only constant time longer then the shortest one will be used and the congestion will be constant. Consider a straight edge (u, v) between nodes $u = (u_1u_2\cdots u_k, i)$ and $v = (u_1u_2\cdots u_k, i + 1)$, where $\lfloor \frac{i+1}{\log n} \rfloor = \lfloor \frac{i}{\log n} \rfloor + 1$ (i.e. u and v are from different groups). The case of a cross edge can be dealt with analogically. The path from $f(u)$ to $f(v)$ will be described in the terms of left and right shifts $L(\ )$ and $R(\ )$. It will consists of several parts, in each part only edges of the same type (i.e. either corresponding to left shifts, or to right shifts) will appear. For clarity let us recall the starting point of the path:

$$f(u) = \underbrace{u_{m+\log n} u_{m+\log n}}_{doubled} |B| \overbrace{w_1 w_2 \cdots w_{\log n}}^{group} | \overbrace{u_{m+\log n+1} \cdots u_{m+2\log n}}^{memory} \clubsuit$$

$$\clubsuit u_{m+\log n+1} \cdots u_{k-1} u_k | u_1 u_2 \cdots u_m |A| \underbrace{u_{m+1} u_{m+1} \cdots u_{m+\log n-1} u_{m+\log n-1}}_{doubled}$$

Starting in $f(u)$, the path to $f(v)$ goes like this:

1. $L_{2+|B|+2\log n}()$

2. $R_{\log n}(\hat{w}_1 \hat{w}_2 \cdots \hat{w}_{\log n})$, where $\hat{w}_1 \hat{w}_2 \cdots \hat{w}_{\log n} = w_1 w_2 \cdots w_{\log n} + 1$

   $R_{\log n+|B|}(u_{m+(3/2)\log n+1} u_{m+(3/2)\log n+1} \cdots u_{m+2\log n} u_{m+2\log n} B)$

   $R_{\log n}()\ R_{|A|}(A) R_{\log n}()$

3. $L_{\log n}(u_{m+1} u_{m+2} \cdots u_{m+\log n})\ L_{|A|}()$

4. $L_{\log n}(u_{m+\log n+1} u_{m+\log n+1} \cdots u_{m+(3/2)\log n} u_{m+(3/2)\log n})$

   $L_{\log n+|B|+2\log n}()$

5. $R_{\log n}(u_{m+\log n+1} \cdots u_{m+2\log n})$

   $R_{|B|+3\log n}()$

This path is called *long embedded edge.*

The first thing we want to prove about the described embedding is that its load is 1. This follows from the structure of the binary representation of the de Bruijn nodes to which the butterfly nodes are mapped. Consider any de Bruijn node $v = (v_1 \cdots v_n)$. If v would be an image of a butterfly node, then $v_1 \cdots v_n$ must contain a subsequence of $2\log n$ doubled bits that starts on some of bit positions $n - 2\log n + 1, n - 2\log n + 3, \cdots, n + 1 \bmod n$, and is bounded by A from the left and by B from the right. There can be at most one such subsequence. Once this subsequence is located it is possible either to determine the relative group number, the number of the group and thus the level of the original butterfly node, or to recognize that no butterfly node was mapped to v. Thus the mapping of nodes is one-to-one.

We next verify that the front-end path dilation is $O(1)$. Since the dilation of only one edge out of $\log n$ consecutive edges on any front-end path is $O(\log n)$ and the dilation of all other is constant (namely 2), the total front-end path dilation is only constant.

The most difficult part is that concerning congestion. We will not care too much about the size of the constants. Therefore for simplicity it will be proved that node congestion is constant which implies also constant edge congestion. Since the load is constant the short embedded edges cannot produce high congestion. The difficulty is with the long embedded edges.

Again, as in the proof of the constant load, the key is in the structure of the binary representation of the nodes that the long embedded edge is passing through. The paths were designed in such a way that each node (resp. its binary representation) along any of the paths would satisfy the following conditions:

- It contains a subsequence of $2\log n$ or $3\log n$ doubled bits, bounded by A from the left and/or by B from the right, that starts close (i.e. at bit position j, $n-5\log n \leq j$ or $j \leq n+3\log n \bmod n$) to the right end of the n bits.
- All the bits, up to a constant number, of the original butterfly node describing its group and row numbers are conserved in a known fashion in each part of the path.

The subsequence of doubled bits serves as a reference point. Consider a de Bruijn node $v=(v_1\cdots v_n)$. We are interested in how many long embedded edges are passing through it. Locate all subsequences of $2\log n$ or $3\log n$ doubled bits with a left margin A and/or right margin B, that start (from left) at bit position $j, n-5\log n \leq j$ or $j \leq n+3\log n \bmod n$. There are at most $4=O(1)$ such subsequences, thanks to margins A and B. Consider one of these subsequences. Since the position of the doubled bits differs in each step in any of the four parts of the long embedded edge, the node v could be passed through at most $2\times 4=O(1)$ times (this is a very rough estimation: 4 is the number of parts of long embedded edges, and there are two types of such edges according to whether the original edge was straight or cross) with the considered subsequence as the real reference point. In total, it was passed through by only constant number of long embedded edges. As noticed earlier, the number of short embedded edges passing through v is also only constant. □

Corollary 33 *The n-dimensional de Bruijn network is $(\frac{1}{2^6}2^n/n^3)$-rearrangeable network, and $(\frac{1}{2^6}2^n/n^4)$-widely nonblocking network, if $O(1)$ congestion of the paths is allowed.*

Proof: Again, according to the Theorem 32 embed the $(n-3\lceil\log n\rceil-6)$-dimensional Beneš network into n-dimensional de Bruijn network and apply the results known for the Beneš network. □

With some effort it is possible to increase the dimension of the embedded Beneš network while retaining the properties of the embedding:

Theorem 34 *For arbitrary constant $\bar{c} > 0$ it is possible to embed an $(n-(1+\bar{c})\log n-6)$-dimensional butterfly into an n-dimensional de Bruijn graph with load 1, congestion $O(1)$ and front-end path dilation $O(1)$.*

Proof: The idea is the same as in the Theorem 32. The difference is in the size of the groups. The size $\log n$ is changed to $c\log n$, $c=\bar{c}/2$, and the mapping of the nodes and edges is changed appropriately. □

This yields better bounds on the nonblocking properties of the de Bruijn.

Corollary 35 *For arbitrary constant $\bar{c} \geq 0$, the n-dimensional de Bruijn network is $(\frac{1}{2^6}2^n/n^{1+\bar{c}})$-rearrangeable network, and $(\frac{1}{2^6}2^n/n^{2+\bar{c}})$-widely nonblocking network, if $O(1)$ congestion of the paths is allowed.*

The result about the rearrangeability almost matches the bound of Theorem 28, up to the factor $1/n^{\bar{c}}$, and the constant difference in congestion.

5.3 Division of de Bruijn Graph into Connected Regions

Before describing the on-line algorithm for the maximum disjoint paths problem, two technical lemmas about the structure of the de Bruijn graph will be given that will be useful for the algorithm.

Let $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_l$, for $l = 2^n/(2^6 n^3)$, denote the de Bruijn nodes, to which the 0–level nodes of $k = (n - 3\log n - 6)$–dimensional butterfly are mapped by the mapping described in Theorem 32. Specifically let

$$\mathbf{v}_i = u_1u_1u_2u_2\cdots u_{\log n}u_{\log n}|B|\overbrace{00\cdots 0}^{\log n}|u_{\log n+1}\cdots u_{2\log n}|u_{\log n+1}u_{\log n+2}\cdots u_k|A\ ,$$

where $u_1u_2\cdots u_k$ is the binary representation of $i-1$. This is the image of $(u_1u_2\cdots u_k, 0)$. For $1 \le i \le l$ let

$$\bar{\mathbf{v}}_i = A|u_1u_1u_2u_2\cdots u_{\log n}u_{\log n}|\overbrace{00\cdots 0}^{\log n}|B|u_1u_2\cdots u_k\ ,$$

where $u_1u_2\cdots u_k$ is the binary representation of $i-1$. Let $R = \{\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2, \cdots, \bar{\mathbf{v}}_l\}$. The set R will be called the set of roots.

Lemma 36 *It is possible to connect the l pairs $(\mathbf{v}_i, \bar{\mathbf{v}}_i)$ of vertices by l paths in such a way, that the maximal edge congestion of these paths is constant.*

Proof: For the connection between $\mathbf{v}_i$ and $\bar{\mathbf{v}}_i$ we use the following path:

1. $L_{|B|+4\log n}()$

2.a $R_{\log n}(u_1u_2\cdots u_{\log n})$

b $R_{|B|+\log n}(00\cdots 0B)$

c $R_{|A|+2\log n}()$

The part 2.b of the path is almost exclusively right rotation with a small change in position of the mark B. By a similar argument as in Theorem 32 it is possible to show that the l paths have only a constant congestion. Again, the key is in the structure of the binary representation of the nodes that the path is passing through. Each node along any of the paths satisfies the following conditions:

- It contains a subsequence of $2\log n$ doubled bits bounded by A from the left that starts close (i.e. at bit position j, $n - 4\log n - |B| \le j$ or $j \le n + |A| + 1 \bmod n$) to the right end of the n bits.
- All the bits describing the level of the original butterfly node (i.e. $u_1u_2\cdots u_n$) are conserved in a known fashion in each part of the path.

From exactly the same reasons as in the proof of Theorem 32, at most constant number of these l paths will meet in a de Bruijn node. □

Lemma 37 *It is possible to divide the nodes of the n–dimensional de Bruijn graph into $l = 2^n/(2^6 n^3)$ disjoint groups $V_1, V_2, \cdots, V_l$, such that, for $1 \leq i \leq l$,*

- $|V_i| \leq 2^7 n^3 - 1$, *and*
- $\bar{\mathbf{v}}_i \in V_i$,
- *for each $\mathbf{w} \in V_i$, all nodes on the path p_w from $\mathbf{w}$ to $\mathbf{v}_i$ corresponding to the operations $R(\ )$ only, belong to the group V_i.*

Proof: Roughly, the group V_i consists of all nodes, to which there is a path from $\bar{\mathbf{v}}_i$ of length at most $3 \log n + 6$, corresponding to left operations $L(\ )$ only ($6 = |A| + |B|$). This set of nodes resembles a binary tree with root $\bar{\mathbf{v}}_i$.

To be more precise, the group V_i is defined as follows, for $1 \leq i \leq l$:

$$V_i = \{\mathbf{w} | d_L(\bar{\mathbf{v}}_i, \mathbf{w}) \leq d_L(\bar{\mathbf{v}}_j, \mathbf{w}) \text{ for all } j \neq i\}\ .$$

The definition says that a node $\mathbf{w}$ belongs to the group of the closest node $\bar{\mathbf{v}}_i$, measured in the $d_R(\ )$ distance from $\mathbf{w}$ (recall that $d_R(\mathbf{w}, \bar{\mathbf{v}}_i) = d_L(\bar{\mathbf{v}}_i, \mathbf{w})$). Since any two nodes from the set of roots R differ in at least one of the $k = n - 3 \log n - 6$ rightmost bits, the closest $\bar{\mathbf{v}}_i$ to any $\mathbf{w}$ is given uniquely. This implies the disjointness of the groups.

Given a node $\mathbf{w} = w_1 w_2 \cdots w_n$, let i be the integer such that the binary representation of $i - 1$ is $w_1 w_2 \cdots w_k$. Then $d_L(\bar{\mathbf{v}}_i, \mathbf{w}) \leq 3 \log n + 6$. That is each node is at distance at most $3 \log n + 6$ from some of the roots R. This yields the bound on the size of $|V_i|$: the number of nodes accessible from $\bar{\mathbf{v}}_i$ along *left* paths of length at most $3 \log n + 6$ is at most $2^7 n^3 - 1$.

Suppose, by contradiction, there is a group V_i and a node $\mathbf{w} \in V_i$ such that the path p_w passes through a vertex $\mathbf{u} \in V_j$ for $j \neq i$. Then $d_L(\mathbf{v}_j, \mathbf{u}) < d_L(\mathbf{v}_i, \mathbf{u})$, and this implies $d_L(\mathbf{v}_j, \mathbf{w}) < d_L(\mathbf{v}_i, \mathbf{w})$. That is, $\mathbf{v}_i$ is not the closest root to $\mathbf{w}$ in the $d_R(\ )$ distance, which is a contradiction with $\mathbf{w} \in V_i$. □

In the next section $\mathbf{v}_i$ will be called the *head* and $\bar{\mathbf{v}}_i$ the *root* of the group V_i.

To get a better idea about the size and shape of the groups, we define

$$\bar{V}_i = \{\mathbf{w} | d_L(\bar{\mathbf{v}}_i, \mathbf{w}) \leq 3\log n + 2\}$$

and prove that $\bar{V}_i \subseteq V_i$. Suppose, by contradiction, there is a node $\mathbf{w} = w_1 w_2 \cdots w_n$ such that $\mathbf{w} \in \bar{V}_i$ but $\mathbf{w} \notin \bar{V}_i$, i.e. $\mathbf{w} \in V_j$ for some $j \neq i$ and $d_L(\bar{\mathbf{v}}_\mathbf{j}, \mathbf{w}) < \mathbf{d_L}(\bar{\mathbf{v}}_\mathbf{i}, \mathbf{w})$. Note that $w_l w_{l+1} w_{l+2} = B$ and $w_{l'} w_{l'+1} w_{l'+2} = B$ for $l = 3\log n + 4 - d_L(\bar{\mathbf{v}}_i, \mathbf{w})$ and $l' = 3\log n + 4 - d_L(\bar{\mathbf{v}}_\mathbf{j}, \mathbf{w})$. Therefore $w_1 w_2 \cdots w_{l'-1}$ must contain as a substring $0B$ (recall that $B = 010$). But at the same time, $w_1 w_2 \cdots w_{l'-1}$ is formed of doubled bits (with possible exception of the first one) followed by $\log n$ zeros only, or by zeros only. In both cases this is a contradiction.

It is possible to easily modify these lemmas for the improved embedding of Theorem 34. For a fixed $c > 0$, $l = 2^n/(2^6 n^{1+2c})$ and $k = n - (1+2c)\log n - 6$, let

$$\mathbf{v}_i = u_1 u_1 u_2 u_2 \cdots u_{c\log n} u_{c\log n} | B | \overbrace{00\cdots0}^{\log n} | u_{c\log n+1} \cdots u_{2c\log n} | \clubsuit$$
$$\clubsuit u_{c\log n+1} u_{c\log n+2} \cdots u_k | A \ ,$$

and

$$\bar{\mathbf{v}}_i = A | u_1 u_1 u_2 u_2 \cdots u_{c\log n} u_{c\log n} | \overbrace{00\cdots0}^{\log n} | B | u_1 u_2 \cdots u_k \ ,$$

where $u_1 u_2 \cdots u_k$ is the binary representation of $i - 1$.

Lemma 38 *It is possible to connect the l pairs of vertices, $(\mathbf{v}_i, \bar{\mathbf{v}}_i)$, by l paths in such a way, that the maximal edge congestion of these paths is constant.*

Proof: The proof is analogical to the proof of Lemma 36. □

Lemma 39 *It is possible to divide the nodes of the n–dimensional de Bruijn graph into $l = 2^n/(2^6 n^{1+2c})$ disjoint groups $V_1, V_2, \cdots, V_l$, such that, for $1 \leq i \leq l$,*

- $|V_i| \leq 2^7 n^{1+2c} - 1$, *and*
- $\bar{\mathbf{v}}_i \in V_i$,
- *for each $\mathbf{w} \in V_i$, all nodes on the path p_w from $\mathbf{w}$ to $\mathbf{v}_i$ corresponding to the operations $R(\ )$ only, belong to the group V_i.*

Proof: Again, the proof is analogical to the proof of Lemma 37. □

5.4 Call Admission and Circuit Routing

Theorem 40 *There is a deterministic $O(n^4)$-competitive algorithm for permanent call problem on the n-dimensional de Bruijn graph, with allowed constant congestion.*

Proof: The algorithm is based on the results obtained in the previous two sections and on the properties of the Beneš network. The algorithm simulates the behavior of the algorithm for Beneš network from Theorem 10.

There are two levels of decision about the requests. Only if the request passes both of them it is accepted. For a node a let $V(a)$ denote the group that a is part of, and let $h(a)$ denote the head of the group. Let ALG denote the algorithm of Theorem 10 for $(n-3\log n-6)$–dimensional Beneš network. Consider a request (a,b) to connect nodes a and b. If some other request from or to $V(a)$ or $V(b)$ was already accepted, then (a,b) is rejected. Otherwise, request $(h(a),h(b))$ is presented to the algorithm ALG for the Beneš network, and (a,b) is accepted by our algorithm if $(h(a),h(b))$ is accepted by ALG. The path between a and b will have several parts. The first part is from a to the root of the group $V(a)$ via a path corresponding to the operation $R(\ )$ only. By Lemma 37 all nodes on this part belong to the set $V(a)$. The second part is from the root of the group to the head of the group. For this we use the path described in Lemma 36. The third part is from the head $h(a)$ to the head $h(b)$. Here we simulate the path of ALG on the Beneš network embedded into the de Bruijn network according to Theorem 32. To connect the head and root of $V(b)$ we use again Lemma 36. The last part is from the root $V(b)$ to b for which the path corresponding to the operation $L(\ )$ only is used. The constant congestion of the embedded Beneš network and of the paths connecting the roots and heads of groups, and the connectivity of the groups ensure the constant congestion of the algorithm.

At least a fraction $1/(2^8 n^3) = \Omega(1/n^3)$ of requests that are accepted by the optimal off-line algorithm proceeds to the second level of decision. The competitive ratio of ALG is $O(n)$. In total, the competitive ratio of our algorithm is at worst $O(n^4)$. $\Box$

At the expense of a higher power of n in the competitive ratio it is possible, according to the Lemma 2, to modify the algorithm to establish the paths in edge disjoint fashion.

Theorem 41 *If the paths are required to be edge disjoint, then there is a* *deterministic $O(n^5)$-competitive algorithm for permanent call problem on the n-dimensional de Bruijn graph.*

Proof: Notice that all paths that the algorithm of Theorem 40 uses are of length $O(n)$. The rest follows from Lemma 2. □

By using Theorem 34, Lemma 38 and Lemma 39 instead of Theorem 32, Lemma 36 and Lemma 37 in the construction of the algorithm in Theorem 40 we get the following theorem.

Theorem 42 *For arbitrary constant $\bar{c} > 0$ there exists a deterministic $O(n^{2+\bar{c}})$-competitive algorithm for permanent call problem on the n-dimensional de Bruijn graph, with allowed constant congestion.*

Again, it can be modified for the case of edge disjoint paths with help of Lemma 2.

Theorem 43 *If the paths are required to be edge disjoint, then for arbitrary constant $\bar{c} > 0$ there exists a deterministic $O(n^{3+\bar{c}})$-competitive algorithm for permanent call problem on the n-dimensional de Bruijn graph.*

Chapter 6

Open Problems and Conclusion

The focus of this work has been on two problems. First, to examine non-blocking properties of several hypercube-like networks. Second, to search for on-line algorithms for the problem of maximum disjoint paths on the same topologies.

For the Beneš network, upper and lower bounds on its nonblocking property were given: the n–dimensional Beneš network B_n is a strictly nonblocking network of order $2^n/n$, but not of order s for any $s > \frac{3}{4}2^n$. The gap between the lower and upper bound is still rather large. It is an open problem what is the exact bound. For small dimensional Beneš networks it was shown that the wide and weak sense nonblocking properties are stronger than the strict sense nonblocking property (B_3 is widely nonblocking network of the order 2^3, but not strictly nonblocking of this order). Is it the same case for all higher dimensional Beneš networks too? Is it possible to extend the algorithm of Theorem 17 for higher dimensions?

The results for the hypercube are derived from the similarities with the Beneš network. However, there are also differences. On the Beneš network the order is more-less given, in which the bits, in which source and sink of a request differ, have to be corrected. This is not the case on the hypercube, where, due to its higher degree there is much more freedom in a choice for a path. This "advantage" (which may be also an advantage for the optimal algorithm) was not exploited in our results. All our paths on the hypercube simulate paths on the Beneš network. The consequence of this strategy is that from *strictly* nonblocking property of the Beneš network a *widely* nonblocking property of the de Bruijn graph is derived only. Nevertheless, arguments for not dealing with the strict sense nonblocking property of the hypercube (and the de Bruijn graph) are given. Is it possible to improve the estimate of the wide sense nonblocking property?

Other open problem for the hypercube is the old question posed by Szymanski: is the directed hypercube (i.e. two neighboring nodes are connected by two directed edges, one in each direction) rearrangeable? It is known it is not along shortest paths.

The nonblocking property of the de Bruijn graph is obtained again from the results for the Beneš network. To apply the Beneš results to the de Bruijn graph an embedding of the Beneš network into the de Bruijn graph was needed. The described embedding is interesting in its own respect. A naturally arising question is whether it is possible to embed a higher dimensional Beneš network into the de Bruijn graph with the same (or better) bounds on load, congestion, dilation and front-end path dilation? For the nonblocking results of the de Bruijn graph the requirements on the paths in the network were slightly weaken: instead of insisting on edge disjointness of the paths a constant congestion was allowed. Is it possible to obtain the same results even when insisting on the edge disjointness? The presented lower and upper bounds on its rearrangeability almost match each other: for any $c > 0$ the graph is not $(\log_2 3 + c)2^{n+1}/$-rearrangeable (via disjoint paths), and it is k-rearrangeable for $k = 2^n/n^{1+c}$, where c is an arbitrary constant $c > 0$ (if constant congestion is allowed). The graph is also shown to be $(\frac{1}{2^6}2^n/n^{2+c})$-widely nonblocking, for arbitrary constant $c \geq 0$ (again, if constant congestion is allowed).

Let us turn our attention to the maximum disjoint paths problem. For the Beneš network and the hypercube two different ways for obtaining an algorithm were used, resulting in the same (up to a constant) competitive ratios. The first one is the more difficult one and it is based on the results about strictly (resp. widely or weakly, but the present results are the same) nonblocking properties of the networks. This yields randomized algorithms. The other way is surprisingly simple, namely "greedy" algorithms. Their good behavior follows from the limited length of paths in the optimal off-line solution on the Beneš network, and in the optimal solution of permutation routing on the hypercube. For the hypercube, we have to pay a cost (a higher power of n in the competitive ratio) for transferring the results from *permutation* routing only to *any* routing (i.e. with possible multiple use of the same nodes). Unfortunately we do not know anything about the properties of the optimal off-line solution on the hypercube when multiple requests from and to the same nodes are allowed. This raises up an interesting question:

are there constants α and c such that given any set of connection requests on the n–dimensional hypercube, there exists an α–approximating solution that uses paths of length at most cn only? There is no such restriction on the length of paths in the optimal solution. A *yes* answer would yield, by Lemma 1, a deterministic on-line $O(n)$-competitive algorithm. Intuitively, the number of too long paths cannot be too high, but we do not know the answer to this problem yet.

Concerning the Beneš network and the hypercube, the greedy algorithms are superior, due to their simplicity and determinism, over those based on the strictly nonblocking properties. However, for the Beneš network the bound on the performance of the described greedy algorithm is tight (i.e. the *described* algorithm surely does not have better competitive ratio). On the other hand there is still a hope for an improvement of the bound of the strictly nonblocking property of the Beneš network. Moreover, any improvement of the bound of the wide or weak sense nonblocking property only, would yield a better algorithm for the Beneš network, and in turn also for the hypercube.

For the de Bruijn graph, all the presented on-line algorithms are based on the embedding of the Beneš network into the de Bruijn graph. This reflects the situation that there are no known results about the properties of the optimal (or approximate) off-line solution for the maximum disjoint paths problem on this graph. It may be an interesting problem to bound the competitive ratio of a greedy algorithm (i.e. accept a request only if there is a free path for it of length at most c times the diameter, where c is a fixed constant) on the de Bruijn (and on other networks with small diameter and high bisection width). For which set of graphs does the greedy algorithm achieve polylogarithmic competitive ratio?

Because of the close correspondence between the Beneš network and the cube connected cycles (e.g. [36, p.449]), and between the de Bruijn and shuffle exchange graphs (e.g. [36, p.481]), the algorithms and other results can be extended also for other hypercube-like topologies. It is worth mentioning that in contrast to simple networks like line and meshes, the deterministic algorithms perform well on the hypercube-like topologies.

We will conclude with two other open problems. The presented randomized algorithms do not beat the competitive ratios of the deterministic ones - how would you design randomized algorithms that will outperform the deterministic ones? Are the lower bounds for deterministic algorithms on the considered topologies higher that the randomized lower bounds?

Bibliography

[1] S. Arora, F. T. Leighton, and B. M. Maggs. On-line algorithms for path selection in a nonblocking network. *SIAM Journal on Computing*, 25(3):600–625, June 1996.

[2] Y. Aumann and Y. Rabani. Improved bounds for all optical routing. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 567–576, New York, NY, USA, Jan. 1995. ACM Press.

[3] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. In J. Díaz and M. Serna, editors, *Proceedings of the 4th Annual European Symposium on Algorithms*, volume 1136 of *Lecture Notes in Computer Science*, pages 431–444, Barcelona, Spain, 25–27 Sept. 1996. Springer.

[4] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 32–40, Palo Alto, California, 3–5 Nov. 1993. IEEE.

[5] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. Manuscript, 1993.

[6] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.

[7] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 412–423, 1994.

[8] Y. Bartal, A. Fiat, and S. Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 531–540, Philadelphia, Pennsylvania, 22–24 May 1996.

[9] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problemy Peredachi Informatsii (in Russian)*, 9:84–87, 1973.

[10] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission (translated from Problemy Peredachi Informatsii)*, 9:64–66, 1974.

[11] B. Beizer. The analysis and synthesis of signal switching networks. In *Proceedings of the Symposium on Mathematical Theory of Automata*, pages 563–576. Brooklyn, NY, Brooklyn Polytechnic Institute, 1962.

[12] S. Ben-David, A. Borodin, G. T. R. Karp, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

[13] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 379–386, Baltimore, Maryland, 14–16 May 1990.

[14] V. E. Beneš. Heuristic remarks and mathematical problems regarding the theory of connecting-systems. *Bell System Technical Journal*, 41, 4:1201–1248, 1962.

[15] V. E. Beneš. Permutation groups, complexes and rearrangeable connecting network. *The Bell System Technical Journal*, 43, 4:1619–1640, 1964.

[16] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[17] N. G. D. Bruijn. A Combinatorial Problem. *Koninklijke Nederlandsche Akademie van Wetenschappen Proc.*, 49:758–764, 1946.

[18] D. G. Cantor. On non-blocking switching networks. *Networks*, 1:367–377, 1972.

[19] S. B. Choi and A. K. Somani. Rearrangeable circuit-switched hypercube architectures for routing permutations. *Journal of Parallel and Distributed Computing*, 19(2):125–130, Oct. 1993.

[20] R. Cypher, F. M. auf der Heide, C. Scheideler, and B. Vöcking. Universal algorithms for store-and-forward and wormhole routing. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 356–365, Philadelphia, Pennsylvania, 22–24 May 1996.

[21] P. Feldman, J. Friedman, and N. Pippenger. Wide-sense nonblocking networks. *SIAM Journal on Discrete Mathematics*, 1, 1988.

[22] S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 563–572, Pittsburgh, PN, Oct. 1992. IEEE Computer Society Press.

[23] S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. *IEEE Transactions on Computers*, 45, 1996.

[24] G. Fertin. Routing permutations in the hypercube of dimension 4. Manuscript, LaBRI, Bordeaux, France, 1997.

[25] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proceedings of the 2nd Israeli Symposium on Theory of Computing and Systems*, pages 285–293, 1993.

[26] M. Grammatikakis, D. Hsu, M. Kraetzl, and J. Sibeyn. Packet routing in fixed-connection networks: A survey. *Journal of Parallel and Distributed Computing*, Accepted.

[27] Q.-P. Gu and H. Tamaki. Routing a permutation in the hypercube by two sets of edge-disjoint paths. In *IPPS: 10th International Parallel Processing Symposium*. IEEE Computer Society Press, 1996.

[28] M. C. Heydemann, J. Opatrny, and D. Sotteau. Embeddings of hypercubes and grids into de Bruijn graphs. *Journal of Parallel and Distributed Computing*, 23(1):104–111, Oct. 1994.

[29] Juan A. Garay and Inder S. Gopal and Shay Kutten and Yishay Mansour and Moti Yung . Efficient on-line call control algorithms. *J. Algorithms*, 23(1):180–194, 1997.

[30] R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, (9):45–68, 1975.

[31] J. Kleinberg. *Approximation Algorithms for Disjoint Paths Problems.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.

[32] J. Kleinberg and R. Rubinfeld. Short paths in expander graphs. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 86–95, 1996.

[33] J. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, pages 26–35, 1995.

[34] J. Kleinberg and É. Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 52–61, Los Alamitos, Oct. 1995. IEEE Computer Society Press.

[35] P. Kolman. On nonblocking properties of the Beneš network. In *Proceedings of the 6th Annual European Symposium on Algorithms*, pages 259–270, Aug. 1998.

[36] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes.* Morgan Kaufmann, San Mateo, 1992.

[37] S. Leonardi. On-line network routing. In G. J. W. Amos Fiat, editor, *Online Alogithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 242–267. Springer, 1998.

[38] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén. Online randomized call control revisited. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 323–332, San Francisco, California, 25–27 Jan. 1998.

[39] R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.

[40] A. Lubiw. Counterexample to a conjecture of szymanski on hypercube routing. *Information Processing Letters*, 35(2):57–61, June 1990.

[41] G. A. Margulis. Explicit constructions of expanders. *Problemy Peredachi Informatsii (in Russian)*, 9:71–80, 1973.

[42] G. A. Margulis. Explicit constructions of expanders. *Problems of Information Transmission (translated from Problemy Peredachi Informatsii)*, 9:325–332, 1975.

[43] R. Melen and J. S. Turner. Nonblocking multirate networks. *SIAM Journal on Computing*, 18(2):301–313, 1989.

[44] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, Feb. 1993.

[45] N. Pippenger. Telephone switching networks. In *AMS Proc. Symposia on Applied Mathematics*, volume 26, pages 101–133, 1982.

[46] C. E. Shannon. Memory requirements in a telephone exchange. *Bell Syst. Tech. Journal*, 29:343–349, 1950.

[47] G. Siegmund. *ATM - Die Technik des Beitband-ISDN*. Huethig, Heidelberg, 1994.

[48] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, Feb. 1985.

[49] T. Szymanski. On the permutation capability of a circuit-switched hypercube. In *International Conference on Parallel Processing, Vol.1: Architecture*, pages 103–110, Pennsylvania, USA, Aug. 1989. The Pennsylvania State University.

[50] M. Tompa. Lecture notes on message routing in parallel machines. Technical Report TR-94-06-05, University of Washington, Department of Computer Science and Engineering, June 1994.

[51] J. Turner and N. Yamanaka. Architectural choices in large scale ATM switches. Technical Report WUCS-97-21, Department of Computer Science, Washington University Saint Louis, 1997.

[52] A. Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, Jan. 1968.