Short Disjoint Paths on Hypercubic Graphs[*]

Petr Kolman
Heinz Nixdorf Institute, Paderborn, Germany
Charles University, Prague, Czech Republic
`kolman@kam.ms.mff.cuni.cz`

January 25, 2000

Abstract

The main problem addressed in this paper is the on-line version of maximum disjoint paths problem (MDPP) on hypercubic graphs. It is defined as follows. For a given graph there is a sequence of requests, i.e. pairs of nodes. For each of the requests the algorithm has to decide, without knowledge of the future requests, whether to reject it or to accept it, and for the accepted request to provide a path between its terminal nodes, that is disjoint with all the paths established previously. The aim is to accept in each sequence as many requests as possible. For comparing quality of different algorithms competitive analysis is used. The on-line MDPP was already considered on several other topologies like line, tree, mesh and expander graphs. Since there is no hope for general algorithm performing well on all graphs it is reasonable to deal with the problem on specific graphs or classes of graphs.

The fundamental network routing problem arises in several practical applications, including telephone switching in telecommunications networks, communication between processors in large scale parallel computers, switching in high-speed networks or communication in optical networks.

The bounded greedy algorithm is considered in this paper: it accepts a request whenever there is a free path for it that is "sufficiently" short. Kleinberg and Rubinfeld proved this algorithm to have polylogarithmic competitive ratio on bounded degree expanders. The main contribution of this paper is a proof that the same algorithm achieves polylogarithmic competitive ratio on hypercubic networks as well, namely $O(\log N)$ on the Beneš network (when used as an indirect network), $O(\log^2 N)$ on the hypercube and $O(\log^{3+c} N)$ on the De Bruijn graph, for an arbitrary $c > 0$. For the analysis on the De Bruijn graph, a new embedding of the butterfly in the De Bruijn graph and a new decomposition of the De Bruijn graph are described. The embedding is of interest of its own. Specifically, for an arbitrary $c > 0$, it is possible to embed an $(n - (1 + c)\log n - 6)$-dimensional butterfly in an n-dimensional De Bruijn graph with load 1, constant congestion, and constant front-end path dilation (which is dilation of input-output paths of the butterfly).

---

[*] This research has been partially supported by GACR under Grant 102/97/1055 and by DFG-GK 124 "Parallele Rechnernetzwerke in der Produktionstechnik".

1 Introduction

The problem of on-line searching for disjoint paths on a graph comes from several different practical applications.

A long time ago this problem emerged in the research of telecommunication networks. Consider a telephone switchboard, represented as a graph. 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.

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.

In the last years the development of network technologies is going on. The high speed of data transmission in high speed networks brings new problems: 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 [5, 6]. 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. Similar problems related to edge disjoint paths appear also in optical routing [1].

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.

To be precise we define our problem formally. The *problem of edge disjoint paths* 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 [15] that this is an NP–complete problem. There is an optimization version of it, called *maximum disjoint paths problem* (MDPP), 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, too. 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, before knowing the next requests in the input sequence, 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. This problem is also called *permanent call problem.* 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 [9].

In this paper the performance of the *bounded greedy algorithm* (BGA) on hypercubic graphs will be examined. The algorithm works as follows. Let L be a suitable chosen parameter (a multiple of the diameter in the case of hypercubic and expander graphs). Given a request, reject it if there is no free path of length at most L between its terminal nodes. Otherwise accept it and for routing use any such path. Kleinberg and Rubinfeld [17] showed that this algorithm achieves polylogarithmic competitive ratio on bounded degree graphs with strong expansion properties. This paper extends this result by proving that the algorithm achieves polylogarithmic competitive ratio on hypercubic networks as well, namely $O(\log N)$ on the Beneš network (when used as an indirect network), $O(\log^2 N)$ on the hypercube and $O(\log^{3+c} N)$ on the De Bruijn graph, for an arbitrary $c > 0$. From similarities between the De Bruijn and the shuffle-exchange graph (cf. [22, p.482]) it follows that the algorithm works on the shuffle-exchange too, with the same competitive ratio.

The bounds on the performance of the algorithm are derived in a straightforward way from the fact that some off-line approximations to the MDPP use short paths only. For the Beneš network and the hypercube we make use of previous results, for the De Bruijn graph a new $O(\log^{2+c} N)$ approximation is given, for an arbitrary $c > 0$. To derive such an approximation, a new embedding of the Beneš network in the De Bruijn graph and a new decomposition of the De Bruijn graph are described. The embedding is of interest of its own. Specifically, for an arbitrary $c > 0$, it is possible to embed an $(n - (1 + c)\log n - 6)$-dimensional Beneš network in an n-dimensional De Bruijn graph with load 1, constant congestion, and constant front-end path dilation (which is dilation of input-output paths of the Beneš network).

The rest of this section gives a brief overview of previous work on the problem of disjoint paths and also necessary network definitions. Section 2 gives the basic theorem that relates the performance of the bounded greedy algorithm to properties of an off-line approximation and demonstrates its use on the Beneš network and the hypercube. In Section 3 the off-line $O(\log^{2+c} N)$ approximation of MDPP on the De Bruijn graph is constructed, for which purpose first the embedding of the butterfly in the De Bruijn, and the decomposition of the De Bruijn graph are described. Section 4 concludes with some open problems.

1.1 Previous work

The on-line MDPP was already considered for several different topologies. Garay, Gopal, Kutten, Mansour and Yung [11, 12] 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 [5]. 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 [4] 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 [6], 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 [18], and it was generalized for a special class of planar graphs ("densely embedded, nearly-Eulerian graphs"). Awerbuch et al [6] dealt also with the hypercube and they gave an $\Omega(\log\log N)$ randomized lower bound for it. Kolman [20] described a randomized $O(\log N)$–competitive algorithm for the Beneš network (when used as an indirect network) and a randomized $O(\log^2 N)$–competitive algorithm for the hypercube.

Most of the mentioned randomized 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 [23] 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 many of these topologies are much higher. For the line network there is a trivial lower bound $\Omega(N)$ (e.g. [3]), which can be easily generalized to $\Omega(d)$ for any diameter d tree. Awerbuch et al [6] mention a deterministic $\Omega(\sqrt{N})$ lower bound for $N \times N$ mesh, by Blum, Fiat, Karloff and Rabani. Kleinberg [16] provides an alternative proof. The known deterministic on-line algorithms for the MDPP, with polylogarithmic competitive ratios, are for expander graphs [17] and for the hex [6].

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 [7] 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 [3]. The authors also gave a matching lower bound for this setting.

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 [2] proposed a general technique that enables to extend results for this problem also for optical routings with multiple wavelengths. Also, 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 with the “classify and randomly select technique” by Lipton and Tomkins [24], cf. [5].

1.2 Network Definitions

The n-dimensional *butterfly* network is defined as follows [22, 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 between (w, i) and (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) of length n. The n-dimensional *Beneš network* B_n consists of two back-to-back butterflies. Overall there are $2n + 1$ levels. The maximal degree is four and the diameter is $2n$. The bisection width of N-node Beneš network is $\theta(N/\log N)$.

The n–dimensional *hypercube* H_n consists of $N = 2^n$ vertices, namely of vertices named by all possible binary strings of length n. There is an edge between u and v if and only if u and v differ in precisely one bit. The total number of edges is $n2^{n-1}$, the degree of all vertices is n and the diameter is also n. The bisection width of the hypercube is $N/2$. The disadvantage of the hypercube is the high, non-constant degree.

The binary n–dimensional *De Bruijn* graph [10] has the same set of vertices as the hypercube, that is all possible strings of n bits. There is an edge between $u = u_1 \cdots u_n$ and $v = v_1 \cdots v_n$ if and only if $v_i = u_{i+1}$ for $1 \leq i < n$, and v_n is either 0 or 1, or $u_i = v_{i+1}$ for $1 \leq i < n$, and u_n is either

0 or 1. That is, there is an edge between two vertices u and v, if either one of them is a left cyclic shift of the other, or one of them is a left cyclic shift of the other with the last bit switched. The number of edges is 2^{n+1}, the degree is 4 in all vertices. The diameter is n and the bisection width is $\theta(N/\log N)$.

Let G and H be two graphs. An *embedding* of the graph G in the graph H is a mapping f of the vertices of G in the vertices of H, together with a mapping g of edges of G in 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. From parameters 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. We introduce also an additional parameter for embeddings of the butterfly. Consider such a mapping. Let p be a path in the butterfly. A dilation of a path p is a ratio of the length of an image of the path p in the other graph, and its original length in the butterfly. The *front-end path dilation* is the maximal path dilation, where the maximum is taken over all shortest paths between first and last level nodes in the butterfly. This definition is motivated by the fact that often only these paths are considered on the butterfly.

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 in the De Bruijn with the load and expansion 1 [14]. Unfortunately, most of them suffer from a large, non-constant congestion. Since we are dealing with edge disjoint paths we would like to have 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. Koch, Leighton, Maggs, Rao, Rosenberg and Schwabe [19] describe an embedding of an N-node butterfly in an $O(N)$-node shuffle exchange network with constant congestion and load and dilation $O(\log N)$. Compared to our embedding of the butterfly in the De Bruijn network, it has a better expansion and a worse dilation.

2 Basic Method

This section states a simple theorem relating the performance of the bounded greedy algorithm to properties of an off-line approximation. Then, the theorem is applied to the Beneš network and the hypercube.

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

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

Let us concentrate now on the Beneš network and the hypercube. It is well known that the Beneš network is rearrangeable [8]: given any set of permutation requests between its first and last level nodes, it is possible to establish edge disjoint paths for all of them. Thus, given *any* set of requests between its first and last level nodes, we can always easily construct the optimal solution. Each of the paths is of length $2n = O(\log N)$.

For the hypercube, we will make use of the following theorem by Gu and Tamaki [13] (cf. [1]):

Theorem 2 (Gu and Tamaki) *Any permutation on the hypercube can be splitted 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).

Given *any* set of requests on the hypercube, let S be a maximal (inclusion) subset of them that forms a (partial) permutation. Since the degree of hypercube nodes is $\log N$, the size of S is at worst $2 \log N$ times smaller than the size of the optimal off-line solution. For S, the Theorem by Gu and Tamaki guarantees a 16–approximation solution. Thus, there is an $O(\log N)$–approximation solution that uses paths of length at most $2n = 2 \log N$.

The following theorem sums up the bounds on the performance of the BGA on the hypercube and the Beneš network:

Theorem 3 *The competitive ratios of the BGD with parameter* $L = 2n$ *on an* n*-dimensional Beneš network and hypercube are* $2n = O(\log N)$ *and* $O(\log^2 N)$*, resp.*

It is worth mentioning that for some graphs the competitive ratio of BGA cannot be better than diameter of these graphs (BGA has to be able to accept request with terminal nodes at distance diameter). On the other hand for some graphs BGA achieves this competitive ratio - e.g. also on trees or meshes, when the parameter L is set to the diameter. However, for these graphs there are randomized algorithms, that perform much better.

3 De Bruijn Graph

3.1 Embedding of a Butterfly in De Bruijn Graph

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 $L_j(x_1 x_2 \cdots x_j)$ and $R_j(x_j \cdots x_2 x_1)$. Thus the path from the previous example will be described as $L_2(01)R(1)$, given the starting node u.

Theorem 4 *It is possible to embed an* $(n - 2\lceil\sqrt{n}\,\rceil)$*-dimensional butterfly in 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 in 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 $(u_1u_2\cdots u_n, i)$, $1 \leq i \leq n$, 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 $\Omega(n)$. 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 $u = (u_1u_2\cdots u_k, i)$ and let $c = \lfloor i/l \rfloor$. Then u is mapped to De Bruijn node

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

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 $v = (v_1v_2\cdots v_n)$. If there is no $1\ C\ 1$ subsequence in $v_1v_2\cdots v_n$, 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. □

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. It is caused by the fact that in the De Bruijn $2\lceil\sqrt{n}\rceil$) out of n bits are used for the information about levels of the original Beneš node, whereas in the original network $\lceil\log n\rceil$ bits only. 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 5 *It is possible to embed an $(n - 3\lceil\log n\rceil - 6)$-dimensional butterfly in 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:

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

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

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 $\hat{w}_1\hat{w}_2\cdots\hat{w}_{\log n} = w_1w_2\cdots w_{\log n} + 1$

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

   $R_{\log n}()\ R_{|A|}(A)R_{\log n}()$
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, $n-5\log n\le j$ or $j\le n+3\log n \bmod n$) 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 $j, n-5\log n\le j$ or $j\le n+3\log n \bmod n$. 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.

If n is not a power of 2 replace $\log n$ by $\lceil\log n\rceil$ in the proof. $\square$

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

Theorem 6 *For arbitrary constant* $\bar{c} > 0$ *it is possible to embed an* $(n-(1+\bar{c})\log n-6)$*-dimensional butterfly in 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 5. 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. □

3.2 Decomposition of De Bruijn Graph into Connected Regions

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 the $k = (n - 3\log n - 6)$–dimensional butterfly are mapped by the mapping described in Theorem 5. 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 $R = \{\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2, \cdots, \bar{\mathbf{v}}_l\}$.

Lemma 7 (Decomposition of De Bruijn graph) *It is possible to divide the nodes of the* n*–dimensional De Bruijn graph into* $l = 2^n/(2^6n^3)$ *disjoint groups* $V_1, V_2, \cdots, V_l$*, such that, for* $1 \le i \le l$*,*

- $|V_i| \le 2^7n^3 - 1$*, 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* $\bar{\mathbf{v}}_i$ *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 (where $6 = |A| + |B|$). This set of nodes resembles a binary tree with root $\bar{\mathbf{v}}_i$.

To be more precise, for any two De Bruijn nodes $\mathbf{u}$ and $\mathbf{v}$, let

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

and similarly let

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

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 $d_R(\mathbf{w},\mathbf{v}) = d_L(\mathbf{v},\mathbf{w})$.

Then, the group V_i is defined as follows, for $1 \le i \le l$:

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

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 nodes in 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 $2^7 n^3 - 1$.

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 R differ in at least one of the $k = n - 3\log n - 6$ rightmost bits and there is a $\bar{\mathbf{v}}_i$ at $d_R()$–distance at most $3\log n + 6$ from $\mathbf{w}$, the closest $\bar{\mathbf{v}}_i$ to any $\mathbf{w}$ is given uniquely. This implies the disjointness of the groups.

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 $\mathbf{w} \in V_i$. □

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

For the construction of the off-line approximation the following lemma will turn up to be useful.

Lemma 8 *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 and each of these paths is of length $O(\log n)$.*

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 $R_{\log n}(u_1 u_2 \cdots u_{\log n})$

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 5 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, $n - 4\log n - |B| \leq j$ or $j \leq n + |A| + 1 \bmod n$) to the right end of the n bits.
- All the bits describing the level of the original butterfly node (i.e. $u_1 u_2 \cdots u_n$) are conserved in a known fashion in each part of the path.

From exactly the same reasons as in the proof of Theorem 5, at most constant number of these l paths will meet in a De Bruijn node. The bound on the length follows from the definition of the paths. □

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

$$\mathbf{v}_i = u_1 u_1 u_2 u_2 \cdots u_{c\log n} u_{c\log n} |B| \overbrace{00 \cdots 0}^{\log n} | u_{c\log n+1} \cdots u_{2c\log n} | \clubsuit$$

$$\clubsuit u_{c\log n+1} u_{c\log n+2} \cdots u_k | A\ ,$$

and

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

where $u_1u_2\cdots u_k$ is the binary representation of $i-1$.

Lemma 9 *It is possible to divide the nodes of the n–dimensional De Bruijn graph into $l = 2^n/(2^6n^{1+2c})$ disjoint groups $V_1, V_2, \cdots, V_l$, such that, for $1 \le i \le l$,*

- $|V_i| \le 2^7n^{1+2c} - 1$, *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.*

Lemma 10 *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.*

3.3 The Approximation of MDPP

Theorem 11 *For each instance of the MDPP on the De Bruijn graph, there is an $O(n^4)$–approximation of the optimal solution, and this approximation uses paths of length $O(n)$ only.*

Proof. Let us denote T the set of requests of the given instance and for i, $1 \le i \le l = 2^n/(2^6n^3)$, let V_i be the group defined in Lemma 7. Let S be a maximal (inclusion) subset of T such that at most two nodes from each V_i are involved in all the chosen requests. Since the sizes of V_i are bounded by $2^7n^3 - 1$ and the node degree is 4, the number of requests in S is at worst 2^9n^3 times smaller that the size of the optimal solution. In the following it will be shown that it is possible to satisfy all the requests in S with paths of length $O(n)$ and congestion $O(1)$. A maximal (inclusion) subset of these paths that contains disjoint paths only is at worst $O(n)$ times smaller then S. Thus, there is an $O(n^4)$–approximation to the given instance of MDPP.

For a node a let $V(a)$ denote the group that a is part of, and let $r(a)$ and $h(a)$ denote the root and the head, resp., of the group. Let $P = \{(h(a), h(b)) \mid (a,b) \in S\}$. Then P contains requests between nodes on the 0-level of the embedded butterfly only, and these requests form a (partial) permutation. Since any permutation can be realized on the Beneš network with edge disjoint paths and since the congestion and front-end path dilation of the embedded butterfly is $O(1)$ (c.f. Theorem 5), all requests in P can be satisfied with paths of length $O(n)$ and congestion $O(1)$ on the De Bruijn graph: Think about the embedded butterfly as about both first and second half of a Beneš network and simulate the paths that would be used on it. What remains is to show how to connect a with $h(a)$ and b with $h(b)$ for each request $(a,b) \in S$. The path from a will go to $r(a)$ first, using the path corresponding to the $R()$ operations only. It follows from the properties of V_i's that such a path is of length at most n and consists of edges from $V(a)$ only. Similarly with path between b and $r(b)$. Finally, to connect all pairs of $r(a)$ with $h(a)$ and $r(b)$ with $h(b)$, the paths from Lemma 8 will be used. To summarize, we have satisfied all requests in S with paths of length $O(n)$ and congestion $O(1)$ only. A maximal subset of disjoint paths only is the desired $O(n^4)$ approximation for the given instance. □

By using Theorem 6, Lemma 9 and Lemma 10 instead of Theorem 5, Lemma 7 and Lemma 8 in the construction of the approximation in the proof of Theorem 11 we get the following improved theorem (remember $N = 2^n$).

Theorem 12 *For an arbitrary constant $\bar{c} > 0$, for each instance of the MDPP on the De Bruijn graph, there is an $O(\log^{2+\bar{c}} N)$-approximation of the optimal solution, and this approximation uses paths of length $O(\log N)$ only.*

By the reasoning given in Section 2 we get also the next result.

Theorem 13 *For an arbitrary constant $c > 0$, the competitive ratio of the BGD, with suitably chosen parameter $L = O(\log N)$, is $O(\log^{3+c} N)$, on the n-dimensional De Bruijn graph.*

4 Open Problems and Conclusion

We have seen that, in contrast to mesh and similar networks, deterministic algorithms perform well on hypercubic networks. In fact, the very simple bounded greedy algorithm performs well on them. Prior to our work, Kleinberg and Rubinfeld proved this algorithm to work well on bounded degree expanders too. For which other graphs or classes of graphs does BGA achieve polylogarithmic competitive ratio? For which graphs is this competitive ratio the best possible for deterministic algorithms? Is it possible to obtain better algorithms with the use of randomization? What are the lower bounds, both deterministic and randomized, for the hypercubic networks?

The bound on the performance of BGA on the hypercube (De Bruijn) was probably too optimistic about the optimal off-line solution. The reason was that we do not know much 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 (De Bruijn), there exists an α–approximating solution that uses paths of length at most cn only? A YES answer would immediately yield an $O(\log N)$ bound on the performance of BGA on the hypercube (De Bruijn). This is also our conjecture that the competitive ratio of BGA on these graphs is better that the presented bounds, namely that it is $O(\log N)$.

Acknowledgments

The author would like to thank Jiří Sgall for many helpful discussions.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] 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.

[12] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *J. Algorithms*, 23(1):180–194, 1997.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] R. R. Koch, F. T. Leighton, B. M. Maggs, S. B. Rao, A. L. Rosenberg, and E. J. Schwabe. Work-preserving emulations of fixed-connection networks. *Journal of the ACM*, 44(1):104–147, Jan. 1997.

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

[21] P. Kolman. *Searching for Edge Disjoint Paths on Hypercube-like Topologies.* PhD thesis, Department of Applied Mathematics, Charles University, Prague, Dec. 1998.

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

[23] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén. On-line 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.

[24] 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.