Migrating and replicating data in networks

Marcin Bienkowski

Computer Science - Research and Development
Computer Science - Research and Development Organ der Fachbereiche Softwaretechnik, Datenbanken und Informationssysteme der Gesellschaft für Informatik e.V. (GI)

ISSN 1865-2034
Volume 27
Number 3

Comput Sci Res Dev (2012) 27:169-179
DOI 10.1007/s00450-011-0150-8

Springer

Your article is published under the Creative Commons Attribution license which allows users to read, copy, distribute and make derivative works, as long as the author of the original work is cited. You may self-archive this article on your own website, an institutional repository or funder's repository and make it publicly available immediately.

Comput Sci Res Dev (2012) 27:169–179
DOI 10.1007/s00450-011-0150-8

SPECIAL ISSUE PAPER

Migrating and replicating data in networks

Marcin Bienkowski

Published online: 12 April 2011
© The Author(s) 2011. This article is published with open access at Springerlink.com

Abstract We survey data management problems in the light of competitive analysis. We review classic results on the file allocation, the page migration, and the page replication problems in a model in which the total cost of communication is subject to optimization. We also briefly present classic and recent extensions of these problems, such as versions that incorporate memory capacity constraints or dynamic changes to the network.

Keywords Online algorithms · Competitive analysis · Data management · File allocation · Page migration

1 Introduction

This survey deals with the problem of managing shared data items among sets of processors. Originally, this issue arose as a memory management problem in a multiprocessor system, in which a globally addressable memory was distributed among a number of local memories. In the modern world, this problem reflects managing shared information in the distributed network of computers. For example, in a distributed program running in the network, nodes usually want to have access to shared data, such as variables, databases, files, etc. Storing all shared data at one or at a few central repositories does not scale well with the increase of the network and is usually not acceptable due to its monetary cost. Therefore, copies of data items have to be stored in the local memories of network nodes. The question how to provide an efficient access to these copies is central to the problems described in this survey.

When a processor wants to read from or write to the shared object, it has to contact a processor holding a copy of the object. Such a transaction incurs a certain cost. In many applications, access patterns to the shared objects change frequently. For example, in parallel pipelined data processing the set of processors accessing a shared variable changes with time. This renders any solution based on a static placement of the shared data inefficient. Thus, to reduce *the total cost of communication*, the strategies described in this paper try to exploit the topological localities of requests and move the shared data, so that the processors accessing it can find it "nearby" in the network.

In this survey, we deal with a *non-uniform model*. It means that shared objects are bigger than the portion that is accessed in a single time step. This is typical when processors in one step want to read or write a single unit of data from a file or a record from a database. On the other hand, to reduce the maintenance overhead, it is assumed that shared objects are indivisible, and can be copied or migrated between nodes only as a whole. This makes object migration and replication much more expensive than a single access to it.

In this survey, we consider mainly a fundamental case, in which there is only one shared file in the network. In this problem, called *file allocation*, copies of the file may be stored in an arbitrary subset of processors. Copies may be created or deleted at runtime, but multiple copies have to be kept consistent. We also concentrate on two complementary subproblems of file allocation.

– The *file migration* problem allows only one copy to be present in the system.

M. Bienkowski (✉)
Institute of Computer Science, University of Wroclaw, Wroclaw, Poland
e-mail: mbi@ii.uni.wroc.pl

– The *file replication* problem arises if the shared file is read-only. Multiple copies are allowed, but no copy can be deleted.

As the primary motivation of these problems came from sharing memory pages in a multiprocessor system, the original names of these problems are *page migration* and *page replication*, respectively.

In reality, the knowledge of future accesses is either partially or completely non-existing. Thus, accesses to shared data can be naturally modeled as an online problem, where the input sequence consists of processor identifiers, which sequentially try to access the shared data. In this survey, we describe work which uses competitive analysis [21] for the data management problems. This approach makes no prior assumptions on the input sequence and compares the cost of the online algorithm to the cost of the optimal offline solution of the same input.

2 Preliminaries

2.1 Network, input, and cost model

A network is modeled as an undirected graph (V, E) consisting of n nodes. Each edge has an associated cost of sending one unit of data over the corresponding communication channel. This cost may represent the load induced by sending data through this communication link. The cost of sending one unit of data between two nodes v_a and v_b, denoted $c(v_a, v_b)$, is defined as the sum of edge costs on the cheapest path between v_a and v_b. It is easy to observe that the function $c(\cdot, \cdot)$ satisfies the triangle inequality.

There is an indivisible shared file of size D, whose copies are stored at nodes of the network. Initially, a single copy is stored at v_1. An input sequence consists of accesses of the form READ(v_i) or WRITE(v_i), meaning that node v_i wants to read or write one unit of data from the file. Node v_i is then called *the requesting node*.

– In case of a read request, v_j contacts the closest (in terms of the cost function) node v_j holding a copy, paying $c(v_i, v_j)$.
– In case of a write request, v_i has to update all the copies, and the cost incurred by such a transaction is the cost of the cheapest Steiner tree [37] containing v_i and all the nodes which hold copies.

After serving the request, the data management algorithm may decide to replicate a copy of the file to another node or delete a copy from the network. Copying incurs a cost $D \cdot c(v_j, v_k)$ if a file is copied from v_j to v_k, whereas deleting a copy is free. Serving one request and optionally reorganizing the placement of the copies constitutes *one step*.

2.2 Competitive analysis

As mentioned above, a data management algorithm has to make its decisions (where to replicate or migrate the copies of the file) after the tth request, exclusively on the basis of the sequence up to request t. To measure the performance of online strategies, the competitive analysis [21] was used. This kind of evaluation, primarily introduced by Sleator and Tarjan [36], compares the cost of an online algorithm to the cost of the optimal offline strategy run on the same input sequence. In the following, we assume that an optimal offline algorithm is denoted by OPT, and for any algorithm ALG, $C_{\text{ALG}}(I)$ denotes the cost of this algorithm on input sequence I. We sometimes omit I if it is clear from the context.

An online deterministic algorithm ALG is *R-competitive* if there exists a constant α, such that for any input sequence I, it holds that

$$C_{\text{ALG}}(I) \leq R \cdot C_{\text{OPT}}(I) + \alpha. \tag{1}$$

The minimum R for which ALG is R-competitive is called the *competitive ratio* of ALG. If $\alpha = 0$, then we call such an algorithm *strictly competitive*.

For a randomized algorithm ALG, we replace its cost in the definition above by its expectation, $\mathbf{E}[C_{\text{ALG}}(I)]$. The expected value is taken over all possible random choices made by ALG. In the randomized case, the power given to the adversary has to be further specified. Following Ben-David et al. [13], we introduce two types of adversaries: *oblivious* and *adaptive*. The latter is properly called *adaptive-online* in the literature. An oblivious adversary has to construct the whole input sequence in advance, not taking into account the random bits used by an algorithm. On the other hand, an adaptive adversary, may choose the next requests after seeing the algorithm's state. In this case, the adversary has to provide its own online (deterministic) algorithm, ADV, which processes the same input in parallel to ALG. In (1), we replace $C_{\text{OPT}}(I)$ by $\mathbf{E}[C_{\text{ADV}}(I)]$. Again, the expectation is taken over all random choices of ALG.

The power of these adversaries can be related as shown in [13]. Let R_{OBL}, R_{AD} be the best competitive ratios for randomized algorithms against oblivious and adaptive adversaries, respectively. Let R_{DET} be the best possible ratio for a deterministic algorithm. Then

$$R_{\text{OBL}} \leq R_{\text{AD}} \leq R_{\text{DET}}. \tag{2}$$

3 Page migration

In this subproblem, introduced by Black and Sleator [20], there is only one copy of the file in the whole network. Thus, we do not differentiate between READs and WRITEs as their

effect is the same: they incur a cost $c(v_i, v_j)$, where v_i is the requesting node and v_j is the node holding the file.

To give the reader better insight into the problem, we consider a toy example: a network consisting of just two nodes v_1 and v_2, where initially the algorithm has the file at node v_1. We take a closer look at two extreme cases.

If the algorithm encounters an input $v_1 v_2 v_1 v_2 \cdots$, then surprisingly it does not have to do anything, although each second request incurs cost $c(v_1, v_2)$. The reason is simple: even the optimal offline algorithm cannot do much in such case, i.e., the optimal solution is to keep the file at a fixed node.

On the other hand, for input $v_2 v_2 v_2 \cdots$ it makes sense to migrate the file to v_2. However, if the algorithm did this too early, it would pay a lot, i.e., $D \cdot c(v_1, v_2)$ and the evil adversary would start to give requests at v_1. The optimal solution for this sequence would be to remain at v_1. To prevent such course of action, the algorithm should use the classic paradigm of online algorithms, which can be formulated as a meta-rule:

> *Change the state of your algorithm only when the cost incurred so far (by not changing the state) is proportional to the cost of such a change.*

This rule is popular especially in the rent-or-buy type of problems, e.g. the ski rental problem [31] and we refer to it as the *rent-or-buy paradigm*. In our case, the algorithm should wait roughly D steps before migrating the file to v_2.

This approach can be generalized to n-node networks: the algorithm should migrate the file to the center of current activity, whereas by "current" we mean the last $\Theta(D)$ rounds. These intuitions are formalized in the following subsections.

3.1 Randomized algorithms

As in the case of many other online problems, the randomized solutions for page migration are easier than their deterministic counterparts.

Westbrook [38] presented a simple algorithm CoinFlip (Cf), which in each round—after serving a request at v^*—migrates the file to v^* with probability $\frac{1}{2D}$. Cf follows the rent-or-buy paradigm, i.e., the expected cost of migration is of the same order of magnitude as the amount spent on serving the request. Moreover, the (expected) position of the file slowly converges to the place where the current activity takes place.

Theorem 1 [38] CoinFlip *is* 3*-competitive* (*even against an adaptive adversary*).

Proof We take any input sequence and we run CoinFlip and the adversary's algorithm Adv on it, comparing their (expected) costs in each step. We define a potential function $\Phi = 3 \cdot D \cdot c(v_{\mathrm{CF}}, v_{\mathrm{ADV}})$, where v_{CF} and v_{ADV} are the nodes holding the files of CoinFlip and Adv, respectively. We show that for any step, it holds that

$$\mathbf{E}[C_{\mathrm{CF}} + \Delta\Phi] \le 3 \cdot C_{\mathrm{ADV}}, \tag{3}$$

where $\Delta\Phi$ is the change in the potential. As Φ is bounded, by summing (3) over all steps, we immediately get the competitiveness result.

We conceptually divide each step into two parts. In the first part, the adversary issues the request, both the algorithm and the adversary serve it, and the algorithm (optionally) migrates its file. In the second part, the adversary optionally migrates its file. We prove that (3) holds, separately, in each part.

Analysis for the first part. Let v^* be the node issuing a request. The algorithm pays $c(v_{\mathrm{CF}}, v^*)$ for serving the request and with probability $\frac{1}{2D}$ it pays $D \cdot c(v_{\mathrm{CF}}, v^*)$ for migrating the file to v^*. Altogether, we obtain $\mathbf{E}[C_{\mathrm{CF}}] = \frac{3}{2} \cdot c(v_{\mathrm{CF}}, v^*)$. The potential may change only if the algorithm migrates its file, and therefore $\mathbf{E}[\Delta\Phi] = \frac{1}{2D} \cdot [3D \cdot c(v^*, v_{\mathrm{ADV}}) - 3D \cdot c(v_{\mathrm{CF}}, v_{\mathrm{ADV}})]$.
Combining these bounds and using the triangle inequality, we get that

$$\begin{aligned}
&\mathbf{E}[C_{\mathrm{CF}} + \Delta\Phi] \\
&\quad = \frac{3}{2} \cdot [c(v_{\mathrm{CF}}, v^*) + c(v^*, v_{\mathrm{ADV}}) - c(v_{\mathrm{CF}}, v_{\mathrm{ADV}})] \\
&\quad \le \frac{3}{2} \cdot [c(v^*, v_{\mathrm{ADV}}) + c(v^*, v_{\mathrm{ADV}})] \\
&\quad = 3 \cdot C_{\mathrm{ADV}}.
\end{aligned}$$

Analysis for the second part. If the adversary moves its file from v_{ADV} to v'_{ADV}, the corresponding cost is $C_{\mathrm{ADV}} = D \cdot c(v_{\mathrm{ADV}}, v'_{\mathrm{ADV}})$. The algorithm pays nothing and the induced change in the potential is

$$\begin{aligned}
\Delta\Phi &= 3 \cdot D \cdot c(v_{\mathrm{CF}}, v'_{\mathrm{ADV}}) - 3 \cdot D \cdot c(v_{\mathrm{CF}}, v_{\mathrm{ADV}}) \\
&\le 3 \cdot D \cdot c(v'_{\mathrm{ADV}}, v_{\mathrm{ADV}}) \\
&= 3 \cdot C_{\mathrm{ADV}}.
\end{aligned}$$

□

The result above is optimal in general networks against adaptive adversaries (see Sect. 3.3). If the adversary is oblivious, then the ratio can be improved to $1 + \phi \approx 2.618$, where ϕ is the golden ratio [38].

This ratio can be further reduced to $2 + \frac{1}{2D}$ (cf. Table 1) if we consider specific network topologies such as trees, hypercubes, meshes [22] or uniform networks [33]. These algorithms are also optimal; the lower bound of $2 + \frac{1}{2D}$ holds even for a graph consisting of two nodes. All the upper and lower bounds for these networks are *work function* based.

Table 1 Results for the page migration problem

	Lower bound		Upper bound	
General graphs				
Deterministic	3.164	[34]	4.086	[12]
Randomized, adaptive adv.	3	[8]	3	[38]
Randomized, oblivious adv.	$2+\frac{1}{2D}$	[22]	2.618	[38]
Particular graphs, randomized alg., oblivious adv.				
Trees, hypercubes, meshes	$2+\frac{1}{2D}$	[22]	$2+\frac{1}{2D}$	[22]
Uniform networks	$2+\frac{1}{2D}$	[22]	$2+\frac{1}{2D}$	[33]

It means that at any step, for any node v, algorithms computes c_v: the cost of the optimal solution which ends with the file at v. Then a migration decision is made on the basis of this information: nodes with low values of c_v hold the file with higher probability than the nodes with high values of c_v.

3.2 Deterministic algorithms

The first deterministic, 7-competitive algorithm was MOVE-TO-MIN, given by Awerbuch et al. [5]. It can be viewed as a rough derandomization of COINFLIP. The algorithm divides an input sequence into chunks consisting of D steps. Within a chunk, it just gathers statistics: let $v_1, v_2, \ldots, v_D$ be the requesting nodes in a chunk. MOVE-TO-MIN migrates the page to node v^*, such that serving these requests from this node would be cheapest, i.e.,

$$v^* = \arg\min_v \sum_{i=1}^{D} c(v, v_i). \tag{4}$$

We call v^* a *gravity center*.

MOVE-TO-MIN was subsequently improved by Bartal et al. [12]. Their MOVE-TO-LOCAL-MIN (MTLM) algorithm uses different chunk length $L = \Theta(D)$ and the gravity center v^* is chosen to minimize the term $\sum_{i=1}^{L} c(v^*, v_i) + \delta \cdot D \cdot c(v_{\mathrm{MTLM}}, v^*)$, where δ is a numerical parameter. The second summand ensures that the migration is not too expensive. An optimal choice of parameters L and δ allows MOVE-TO-LOCAL-MIN to achieve the competitive ratio of 4.086.

3.3 Lower bounds

As already mentioned above, the best-known ratio against oblivious adversaries is $2+\frac{1}{2D}$; it holds even for two node networks [22].

Higher lower bounds can be shown against adaptive adversaries or deterministic algorithms. In particular, Black and Sleator [20], using a result of Karlin et al. [30], showed a lower bound of 3 for the competitive ratio of any deterministic algorithm. Essentially, the same proof works also for randomized algorithms against adaptive adversaries [8, 11]. This implies that the COINFLIP algorithm is optimal against adaptive adversaries. Below we present a deterministic variant of the proof.

Theorem 2 [8, 11, 20, 30] *For any graph consisting of at least two nodes, the competitive ratio of any deterministic algorithm is at least* 3.

Proof We assume that the graph consists only of two nodes v_a and v_b connected by an edge of cost 1.

Fix any deterministic algorithm ALG. We define three algorithms:

1. algorithm A always keeps its file at v_a;
2. algorithm B always keeps its file at v_b;
3. algorithm C keeps its file at the node not holding a file of ALG.

The adversary creates an input sequence, in which the requests are always issued at the node not holding the file of ALG. We observe that for any step, it holds that

$$C_{\mathrm{ALG}} = C_{\mathrm{A}} + C_{\mathrm{B}} + C_{\mathrm{C}}. \tag{5}$$

Clearly, (5) holds for serving requests, as in this case $C_{\mathrm{ALG}} = 1$, $C_{\mathrm{C}} = 0$ and exactly one of C_{A} and C_{B} equals 1. Moreover, algorithms A and B never pay anything for migration, as they never change the position of their files. On the other hand, C migrates its file exactly when ALG does; they both pay D in this case.

Summing (5) over all steps, we may infer that one of the algorithms A, B, C has cost at most one third of ALG, and therefore $C_{\mathrm{OPT}} \le \min\{C_{\mathrm{A}}, C_{\mathrm{B}}, C_{\mathrm{C}}\} \le \frac{1}{3} \cdot C_{\mathrm{ALG}}$. □

For deterministic algorithms, the lower bound was improved to $85/27 \approx 3.148$ by Chrobak et al. [22] and recently to 3.164 by Matsubayashi [34]. Both bounds hold only in special graphs.

4 File allocation

In this general problem, we differentiate between read and write requests. We observe that when only WRITE requests are present, then the problem is equivalent to page migration, as there is no point of having more than one copy of the file. On the other hand, if READ requests dominate in the input sequence, then having many copies may be desirable. It appears that the file allocation problem and its read-only version, page replication, have a close relation to the online Steiner tree problem. Hence, below we present this problem along with a simple solution.

4.1 Online steiner tree

In the Steiner tree problem [37], one has to build a minimum cost tree, called Steiner tree, which contains all nodes from a given set $Q \subseteq V$. In an online version of this problem [4, 29], the nodes of Q are revealed one by one, and the algorithm has to maintain a Steiner tree covering the nodes which already appeared in the input sequence. Moreover, during its runtime, the algorithm may only add nodes and edges to the Steiner tree and is not allowed to remove them.

A natural deterministic online strategy for this problem is a greedy one: upon seeing a new node v, connect it to the current tree T using the cheapest path between v and a node from T.

Theorem 3 [11, 29] *The greedy algorithm for the online Steiner tree problem is* $\lceil \log n \rceil$*-competitive.*

Proof Let Q be a subset of vertices, S be the minimum cost Steiner tree on Q and C_{OPT} its weight. For any $v \in Q$, let $C(v)$ be the cost of the greedy algorithm of connecting v to its Steiner tree. We show the following claim: for any set $B \subseteq Q$, it is possible to choose $B' \subseteq B$, such that $|B'| = \lfloor |B|/2 \rfloor$ and $\sum_{v \in B'} C(v) \leq C_{\mathrm{OPT}}$. By starting from the set of all Steiner vertices and recursively applying this claim, we get that the cost of the greedy algorithm is $\sum_v C(v) \leq \lceil \log n \rceil \cdot C_{\mathrm{OPT}}$.

The claim can be shown as follows. Let H be a cycle created by traversing the whole tree S and returning to the starting point. Its weight equals $2 \cdot C_{\mathrm{OPT}}$. We number all the vertices of B in the order they lie on the cycle by $b_1, b_2, \ldots, b_{|B|}$. It is possible to pick $\lfloor |B|/2 \rfloor$ consecutive pairs, e.g. (b_1, b_2), $(b_3, b_4), \ldots, (b_{|B|-1}, b_{|B|})$, so that the costs within these pairs sum up to at most C_{OPT}. From each such pair (b_i, b_j), we pick the node which appears later in the input sequence, say b_i; B' is the set of these nodes. When connecting such a node b_i to the Steiner tree, b_j is already in the tree, and hence $C(b_i) \leq c(b_i, b_j)$. This implies that $\sum_{v \in B'} C(v) \leq C_{\mathrm{OPT}}$. □

The online Steiner tree problem [4, 29, 39] can be viewed as a version of file allocation, where only READ requests are present, $D = 1$ and the algorithm has to replicate to the requesting node. These two problems are closely related: Bartal et al. [11] proved that the existence of a c-competitive algorithm for the file allocation implies the existence of a strictly c-competitive algorithm for the online Steiner tree. From the lower bound of $\Omega(\log n)$ for the competitiveness of the latter problem [29] (holding for particular topologies), follows the same lower bound for file allocation. This lower bound holds even for randomized algorithms against adaptive adversaries.

4.2 Algorithms for general networks

Bartal et al. [11] developed an algorithm STEINERBASED (SB), which uses a solution to the online Steiner tree problem. Their algorithm is $O(\log n)$-competitive against adaptive adversaries and works as follows.

Let A be an algorithm solving the online Steiner tree problem. In the beginning, A contains just one node, v_1. In each round, SB keeps the copies at nodes covered by A.

Upon seeing a request issued at v_j, SB first serves it. If v_j requests a READ operation, then with probability $1/D$, SB feeds A with a request at v_j and replicates copies to the nodes newly added to the tree of A. If v_j requests a WRITE operation, then with probability $\frac{1}{\sqrt{3}D}$, SB replicates a copy to v_j, removes all the remaining copies, and restarts algorithm A with v_j as its initial configuration.

Theorem 4 [11] *If A is a strictly c-competitive algorithm for the online Steiner tree problem against an adaptive adversary, then* SB *is $((2+\sqrt{3}) \cdot c)$-competitive for the file allocation problem against an adaptive adversary.*

The proof uses a potential function. Interestingly, A is treated completely as a black box, even if the proof of its competitiveness is not potential function based (as it is the case for the greedy algorithm, see Theorem 3). In order to cope with this issue, Bartal et al. [11] showed that any c-competitive online algorithm for a *configuration problem* has a proof of c-competitiveness which uses so-called *natural potential function*. Configuration problems are a large class of all online algorithms; in particular they include the online Steiner tree problem. Theorems 3 and 4, combined, yield an asymptotically optimal $O(\log n)$-competitive algorithm for the file allocation against an adaptive adversary.

The authors of [11] showed also how to get rid of the central control (which is useful for example for locating the nearest copy of the object) and create an $O(\log^4 n)$-competitive algorithm, which works in a distributed fashion. Awerbuch et al. [5] constructed deterministic algorithms (centralized and distributed ones) for file allocation problem attaining asymptotically the same ratios. For oblivious adversaries, this ratio was subsequently improved by Bartal [10] and then by Fakcharoenphol et al. [23] to $O(\log n)$ by approximating a graph by a randomly chosen tree and applying a simple distributed solution for a tree.

4.3 Specific topologies

The competitive ratio of $\Theta(\log n)$ can be improved in common regular topologies like trees or uniform networks (i.e., complete networks with equal costs between each pair of nodes), see Table 2.

The best lower bounds of 3 for deterministic algorithms for these topologies are in fact lower bounds for the page

Table 2 Results for the file allocation problem

	Lower bound		Upper bound	
General networks				
Deterministic	$\Omega(\log n)$	[11]	$O(\log n)$	[5]
Rand., adaptive adv.	$\Omega(\log n)$	[11]	$O(\log n)$	[11]
Rand., oblivious adv.	$\Omega(\log n)$	[11]	$O(\log n)$	[11]
Specific topologies				
Uniform, deterministic	3	[20]	3	[11]
Trees, deterministic	3	[20]	3	[33]
Trees, rand., oblivious	$2+\frac{1}{D}$	[33]	$2+\frac{1}{D}$	[33]

migration problem (see Theorem 2). For uniform networks, a matching upper bound based on counters was presented by Bartal et al. [11]. For trees, Lund et al. [33] used work functions to create optimal algorithms: a 3-competitive deterministic one and a $(2+\frac{1}{D})$-competitive randomized one.

5 Page replication

In this subproblem of file allocation, introduced by Black and Sleator [20], the file is read-only. The algorithm starts with a file at v_1 and may later only replicate it to other nodes. In this problem, we allow only strictly competitive algorithms. If we did not impose such a restriction, an algorithm which replicates the file to all nodes at the very beginning would be 0-competitive, as the cost of such a replication is independent of the request sequence and can be hidden in the additive constant α of (1).

The reduction to the online Steiner tree (cf. Sect. 4.1), works also for the page replication problem, and thus the competitive ratio of any algorithm is $\Omega(\log n)$ (even for randomized algorithms against oblivious adversaries). Therefore, the research concentrated on specific topologies, like uniform graphs, trees, or rings.

5.1 Two node graphs

We start with describing algorithms which work on a two node graph connected by an edge of cost 1. Such a graph we call EDGE.

Theorem 5 [20] *There exists a 2-competitive deterministic algorithm for the page replication problem on* EDGE.

Proof Let ALG keep a count of the number of requests at v_2. When it reaches D, ALG replicates the file to v_2.

Let k be the number of requests at v_2. We observe that $C_{\text{ALG}} = k$ when $k < D$, and $C_{\text{ALG}} = D + D$ otherwise. On the other hand, $C_{\text{OPT}} \geq \min\{k, D\}$. Thus, $C_{\text{ALG}} \leq 2 \cdot C_{\text{OPT}}$. □

Theorem 6 [3] *Let* $\mathrm{e}_D = (1+1/D)^D$. *There exists a* $\frac{\mathrm{e}_D}{\mathrm{e}_D-1}$*-competitive randomized algorithm for the page replication problem on* EDGE (*against an oblivious adversary*).

Proof Algorithm GEOMETRIC first chooses a random number from the set $\{1, 2, \ldots, D\}$, so that i is picked with probability

$$p_i = \alpha \cdot \left(1+\frac{1}{D}\right)^{i-1}, \quad \text{where } \alpha = \frac{1}{D \cdot (\mathrm{e}_D - 1)}. \tag{6}$$

It is easy to verify that $\sum_{i=1}^{D} p_i = 1$. When the number of requests at v_2 reaches this randomly chosen number, then GEOMETRIC replicates the file to v_2.

For the analysis, let k be the number of requests at v_2 and let i be the randomly chosen number. First, we observe that if $k \geq D$, then the costs of GEOMETRIC and OPT do not change when we set $k = D$. Thus, in the following we assume that $k \leq D$. In this case $C_{\text{OPT}} = k$. If $i \leq k$, then $C_{\text{GEOM}} = i + D$, and otherwise $C_{\text{GEOM}} = k$. Thus,

$$\mathbf{E}[C_{\text{GEOM}}] = \sum_{i=1}^{k} p_i \cdot (i+D) + \sum_{i=k+1}^{D} p_i \cdot k. \tag{7}$$

After substituting the values of p_i and a few algebraic transformations, we obtain that

$$\mathbf{E}[C_{\text{GEOM}}] = \frac{\mathrm{e}_D}{\mathrm{e}_D - 1} \cdot k = \frac{\mathrm{e}_D}{\mathrm{e}_D - 1} \cdot C_{\text{OPT}}. \tag{8}$$

We observe that $\frac{\mathrm{e}_D}{\mathrm{e}_D-1} \to \frac{\mathrm{e}}{\mathrm{e}-1} \approx 1.582$ when D tends to infinity. □

For randomized algorithms against adaptive adversaries Albers and Koga [3] gave a memoryless COINFLIP algorithm, which—upon seeing a request at v_2—replicates the file to v_2 with probability $1/D$. By using a standard potential function argument, they showed that the competitive ratio of COINFLIP is 2.

All the algorithms presented in this subsection are optimal; matching lower bounds can be proven in similar manner.

5.2 Trees and uniform networks

The exact competitive ratios for trees and uniform networks are known and given in Table 3. Black and Sleator [20] proposed how to extend the deterministic algorithm for EDGE to any tree preserving the competitive ratio of 2. Their algorithm R-TREE keeps counters for all nodes. When a node v_j is requesting, its counter and the counters of all nodes on the path between v_j and the closest node holding a copy are increased.

Table 3 Results for the page replication problem

	Lower bound		Upper bound	
Trees and uniform networks				
Deterministic	2	[20]	2	[20]
Rand., adaptive adv.	2	[3]	2	[32]
Rand., oblivious adv.	$\frac{e_D}{e_D-1} \to 1.582$	[3]	$\frac{e_D}{e_D-1} \to 1.582$	[3]
Rings				
Deterministic	2.366	[26]	4	[3]
Rand., adaptive	1.75	[26]	4	[3]
Rand., oblivious adv.	1.75	[26]	2.373	[26]

In the analysis, we may employ a *factoring technique*, i.e., we view the behavior of this algorithm from the perspective of a single edge e. Then the algorithm does exactly the same as the algorithm for EDGE: when the counter at one end of e reaches D, the file is replicated across e. In effect, the total cost of the algorithm for a tree is the sum of costs of algorithms for single edges. This implies that the algorithm for a tree is 2-competitive.

Albers and Koga [3] use similar techniques to extend GEOMETRIC and COINFLIP to arbitrary trees, preserving their competitiveness.

For uniform graphs, it is sufficient to observe that, without loss of generality, any algorithm (online or offline) may use only edges (v_1, v_i) for $2 \le i \le n$. Thus, we may apply the algorithms for trees described above.

5.3 Rings

Albers and Koga [3] showed a general technique of transforming a solution for a tree to the solution for a ring. They associate each point from the ring with a real number from $[0, 1)$, where v_1 is mapped to the point 0. Then they cut the ring at the point antipodal to v_1, i.e., $u = 1/2$, obtaining a tree (which is in fact a line). All requests are then treated as if they were issued on this tree. They showed that any c-competitive algorithm for a tree yields a $2 \cdot c$-competitive algorithm for a ring.

By applying this reduction to the upper bounds for trees, we immediately get a 4-competitive deterministic algorithm, a 4-competitive randomized memoryless algorithm against adaptive adversaries, and a 3.164-competitive randomized algorithm against oblivious adversaries [3].

The results on rings were subsequently improved for various special cases by Fleisher, Głazek, and Seiden [25–28]. In particular, they refine the described ring splitting technique. Let the cut point u be chosen randomly with the following probability density function:

$$f(u) = \begin{cases} \frac{1}{2 \cdot u^2} & \text{if } u \ge 1/2, \\ \frac{1}{2 \cdot (1-u)^2} & \text{otherwise.} \end{cases} \tag{9}$$

For any c-competitive algorithm for a tree, this technique yields a randomized algorithm for a ring that is $\frac{3}{2} \cdot c$-competitive against an oblivious adversary [26]. Moreover, the distribution over cut points is optimal. When we use a 1.582-competitive randomized algorithm for trees [3], we immediately obtain a 2.373-competitive randomized algorithm for rings.

This splitting technique is a special case of a more general technique called *probabilistic approximation of a metric space by a tree metric*, in which we randomly embed a more complicated graph into a tree and we solve an original problem on a tree. Embeddings of general graphs are given for example in [9, 10, 23].

Fleischer, Głazek and Seiden [26] also improved lower bounds for rings, see Table 3.

6 Extension: memory constraints

If multiple objects are present in the network and the local memory capacity at nodes is limited, then we cannot run an independent file allocation scheme for each single object in the network. Above all, it is not possible to copy a file into a node's memory if it is already full. Possibly, a copy of another object has to be dropped, which induces problems if it was the last copy present in the network. This leads to a so called *distributed paging* problem, where file allocation solutions have to be combined with schemes known from paging (see e.g. [24, 36]).

For uniform networks, Bartal et al. [11] gave the algorithm Distributed-Flush-When-Full, which is $O(m)$-competitive, where m is the total number of copies that can be stored within the network. They also proved that this bound is tight by showing a lower bound of $\Omega(m)$ for adaptive adversaries. Awerbuch et al. [6] used randomized uni-processor paging algorithms [1, 24, 35] to construct asymptotically optimal algorithm HEAT & DUMP, $O(\max\{\log(m - f), \log k\})$-competitive against an oblivious adversary. In this context, f is the number of different files in the network, and k is the maximum number of files that can be stored at any node. If we again restrict the number of file copies to one, it results in a problem called *page migration with memory constraints*. Albers and Koga [2] presented deterministic and randomized algorithms for this problem, which are much simpler than their distributed paging counterparts, and achieve competitive ratios $O(n)$ and $O(\log n)$, respectively.

For general networks, Awerbuch et al. [7] adopted the *resource augmentation* extension suggested previously for single processor paging. In order to compensate the optimal offline algorithm's advantage of knowing the future, Sleator and Tarjan [36] proposed limiting the memory capacity that the optimal algorithm has at its disposal. This extension,

which goes beyond the pure competitive analysis, allowed the authors of [7] to present a deterministic $O(\text{polylog}\, n)$-competitive algorithm, under the assumption that an online algorithm has $O(\text{polylog}\, n)$ times more memory than the optimal algorithm.

For the case of page migration with memory constraints in general networks, the best algorithm uses probabilistic approximation of metrics by a tree metric. Bartal [9] gave an $O(\log m)$-competitive solution for so-called 2-*hierarchically well-separated trees* (2-HSTs). These are special types of trees in which the edge lengths on the path from the root to leaves decrease at least by the factor of 2. Fakcharoenphol et al. [23] showed that any graph can be (probabilistically) approximated by a distribution over 2-HSTs with a distortion of $O(\log n)$. These two results, combined, yield a randomized algorithm for page migration with memory constraints, which is $O(\log m \cdot \log n)$-competitive in general graphs against an oblivious adversary.

7 Extension: dynamic networks

There are two different types of network dynamics that were considered in this type of research. In the first one, nodes may appear or disappear from the network. In the second one, the cost of communication between nodes may change.

For the former model, the nodes may die or wake up after being unaccessible. These vertex inclusions and exclusions are a part of the input sequence and are dictated by an adversary. It is only assumed that before a node becomes unavailable, the algorithm has enough time to migrate all the copies held by that node. This problem, called DynDFA, was introduced along with a deterministic $O(\text{polylog}(n))$-competitive algorithm by Awerbuch et al. as a part of their general solution for the distributed paging problem [7].

In the remaining part of this section, we focus on the latter type of network dynamics introduced by Bienkowski et al. [17]. We assume that the network is no longer static, but it behaves like a mobile ad-hoc network, i.e., the nodes are allowed to move with a bounded speed. The model of slow changes in the communication costs tries also to capture changes in the available bandwidth in wired networks. Thus, we have to deal with two sources of online events, namely the requests to a file and the movement of the nodes.

7.1 The model

We consider the standard file allocation problem, but with the following addition: before any request, the adversary may additionally move each node changing the corresponding costs of communication (as described below).

Recall that in the standard data management problems, the costs were arbitrary—they only had to fulfill the triangle inequality. Now we modify this notion slightly. Nodes are placed in an unbounded metric space, e.g. Euclidean plane, with the distances between them given by the function d.

Any two nodes are able to communicate directly with each other. The cost of sending a unit of data from node v_a to v_b is defined by a *cost function*

$$c(v_a, v_b) = d(v_a, v_b) + 1 \tag{10}$$

if v_a and v_b are different nodes. The communication within one node is free, i.e., $c(v_a, v_a) = 0$. Essentially, the communication cost is proportional to the distance between these two nodes, plus a constant overhead. This overhead represents the startup cost for establishing connection and is needed for technical reasons (without it no algorithm is able to achieve a finite competitive ratio).

Before issuing a request at a given node, the adversary is allowed to move all nodes, each along the distance of at most 1, and the costs of communication are changed accordingly.

7.2 Hardness results

The considered model may seem too restrictive. However, it appears that even in the simplest scenario, i.e., if the graph consists of just two nodes, the file allocation problem is infeasible and the competitive ratio for the page migration problem is large.

Theorem 7 [19] *Even for a two node graph, no algorithm can be competitive for file allocation in dynamic network.*

Theorem 8 [17] *For migration in dynamic network on a two node graph, the competitive ratio of any algorithm (even randomized one against an oblivious adversary) is at least* $\Omega(\sqrt{D})$.

Proof We concentrate on a proof for a deterministic algorithm Det. We divide input into phases, each of length $D + 2 \cdot \sqrt{D}$ steps. Each phase consists of an *expanding part*, ($\sqrt{D}$ steps), a *main part* (D steps), and a *contracting part* (also $\sqrt{D}$ steps). A phase begins with v_1 and v_2 occupying the same point of the space. Within the expanding part, nodes are moved apart, so that in the tth step of the expanding part $d(v_1, v_2) = t - 1$. Throughout the whole main part, this distance amounts to $\sqrt{D}$. Finally, in the contracting part, nodes are brought closer to each other, so that at the end of the phase they meet again.

In the expanding part, v_1 issues all the requests, and in the contracting one, v_2 is the requesting node. Further, at the beginning of the main part, the adversary looks where Det holds its file and all the requests of the main part are issued at the opposite node.

Table 4 Results for the page migration problem in dynamic networks

	Lower and upper bounds	
Deterministic	$\Theta(\min\{n \cdot \sqrt{D}, D\})$	[17, 18]
Randomized, adaptive adv.	$\Theta(\min\{n \cdot \sqrt{D}, D\})$	[17]
Randomized, oblivious adv.	$\Theta(\min\{\sqrt{D \cdot \log n}, D\})$	[15, 19]

In a single phase P, $C_{\text{OPT}}(P) = O(D)$, as OPT may move to an appropriate node at the beginning of P and pay only for the requests in the expanding or the contracting part. On the other hand, at the beginning of the main part, DET has its file at a "wrong" node, and has essentially two options. If it migrates the file within the main part, it pays $D \cdot \sqrt{D}$. Otherwise, it pays $D \cdot \sqrt{D}$ for serving the requests during this part. Hence, $C_{\text{DET}}(P) \geq D \cdot \sqrt{D} = \Omega(\sqrt{D}) \cdot C_{\text{OPT}}(P)$. This process may be repeated for any number of phases, which yields the lower bound.

The construction above can be extended to any randomized algorithm RAND; at the beginning of the main part, its file is at the "wrong" node with the probability at least $1/2$. Hence, its expected cost is $\mathbf{E}[C_{\text{RAND}}(P)] \geq \frac{1}{2} \cdot D \cdot \sqrt{D} = \Omega(\sqrt{D}) \cdot C_{\text{OPT}}(P)$. □

7.3 General networks

The asymptotic competitive ratios for various types of adversaries for the page migration problem in dynamic networks are given in Table 4.

Lower bounds of $\Omega(\min\{n \cdot \sqrt{D}, D\})$ for adaptive adversaries and $\Omega(\min\{\sqrt{D \cdot \log n}, D\})$ for oblivious ones were given by Bienkowski et al. [17, 19]. They modified the construction of Theorem 8, employing more nodes.

A deterministic $O(D)$-competitive algorithm JUMP [17] always migrates the file to the node which just issued the request. Although it is trivial, it constitutes a building block in all the upper bounds (see Table 4). The remaining algorithms are respectively:

1. a deterministic $O(n \cdot \sqrt{D})$-competitive algorithm MARK;
2. a randomized algorithm EBM, $O(\sqrt{D \cdot \log n})$-competitive against oblivious adversaries.

In the remaining part of this section, we briefly describe algorithm MARK and its analysis.

The algorithm MARK [18] bears some similarities to the algorithm MOVE-TO-MIN (see Sect. 3.2). It also works in chunks, but of length $\sqrt{D}$. This length constitutes a trade-off: it has to be long enough to amortize the movement of the file against the cost of serving the requests in the chunk, and short enough to make the adversarial network changes negligible. Recall that after a chunk, MOVE-TO-MIN moves to a gravity center. However, it appears that in dynamic networks, any algorithm, which considers *only* gravity centers as candidates for storing the file, has no chance to be better than $\Omega(D)$-competitive.

On the other hand, keeping the file close to the gravity center is, generally, a desirable thing. Hence, we consider the following marking scheme, which depends only on the input sequence. Chunks are grouped into epochs, each epoch begins with all nodes unmarked. The first epoch starts with the beginning of the input. In each epoch, we track counters A_i for the part of the epoch seen so far. A_i is the cost of an algorithm, which remains at v_i, and does not move. If such a counter exceeds D, then the corresponding node becomes marked. At the end of a chunk, in which all nodes are already marked, the current epoch ends, the scheme unmarks all nodes, and a new epoch begins.

MARK uses this scheme in the following way. It remains at a node till the end of the chunk, in which this node gets marked, and then moves to any not yet marked node. Additionally, at the end of the last chunk of the epoch, it moves to the gravity center associated with this chunk.

Theorem 9 [18] MARK *is* $O(n \cdot \sqrt{D})$*-competitive.*

Proof (*sketch*) If a node v_i remains far away from the gravity center, A_i increases rapidly, which leads to marking the node. Conversely, if at the end of a chunk a node is not marked, then its distance to the gravity center is at most $O(\sqrt{D})$.

Thus, if MARK migrates the file, it is moved to the neighborhood of the gravity center. The sequence of chunks between two migrations of MARK is called *phase*. Using similar kind of amortized analysis (with adequately chosen potential function) as for the algorithm MOVE-TO-MIN, one may show the following relation for any phase P:

$$C_{\text{MARK}}(P) \leq O(\sqrt{D}) \cdot C_{\text{OPT}}(P) + O(D \cdot \sqrt{D}). \tag{11}$$

We may trade the additive term of $O(D \cdot \sqrt{D})$ for the competitive factor using the properties of the marking scheme. First, the OPT's cost in one epoch is at least D. It follows by the case analysis: if OPT migrates the file, then it is charged at least D, otherwise its file remains at a node v_i, and thus $C_{\text{OPT}}(P) \geq A_i \geq D$. Second, since in each phase at least one new node gets marked, the number of phases in one epoch is at most n. Therefore, the additive terms in one epoch amount to $O(n \cdot D \cdot \sqrt{D})$, which is at most $O(n \cdot \sqrt{D})$ times the optimal cost. This concludes the proof of MARK's competitiveness. □

Straightforward randomization of MARK, i.e., choosing not any, but a random not yet marked node, reduces the number of phases to $\log n$ and the competitive ratio to $O(\sqrt{D} \cdot \log n)$. Choosing different length of chunks and a refined randomization presented in [15] gives an algorithm EBM, which is $O(\sqrt{D \cdot \log n})$-competitive against an oblivious adversary.

7.4 Relaxed models

The competitive ratios of the best possible algorithms for the page migration in dynamic networks are relatively large, even against the weakest, oblivious adversaries. As illustrated in the proof of Theorem 9, the poor performance of algorithms is caused by the fact that the part of the adversary which changes the network and the part which dictates requests may combine and synchronize their efforts. It was therefore proposed that the problem could be analyzed in a more realistic scenario where one of these parts is replaced by a stochastic process [14, 16]. In both cases, the competitive ratios can be greatly decreased.

Acknowledgements This work was partially supported by Ministry of Science and Higher Education (MNiSW) grant number N N206 1723 33, 2007–2010.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

1. Achlioptas D, Chrobak M, Noga J (2000) Competitive analysis of randomized paging algorithms. Theor Comput Sci 234(1–2):203–218
2. Albers S, Koga H (1995) Page migration with limited local memory capacity. In: Proc of the 4th int workshop on algorithms and data structures (WADS), pp 147–158
3. Albers S, Koga H (1998) New on-line algorithms for the page replication problem. J Algorithms 27(1):75–96. Also appeared in Proc of the 4th SWAT, pp 25–36 (1994)
4. Alon N, Azar Y (1992) On-line Steiner trees in the Euclidean plane. In: Proc of the 8th ACM symp on computational geometry (SoCG), pp 337–343
5. Awerbuch B, Bartal Y, Fiat A (1993) Competitive distributed file allocation. In: Proc of the 25th ACM symp on theory of computing (STOC), pp 164–173
6. Awerbuch B, Bartal Y, Fiat A (1993) Heat & Dump: competitive distributed paging. In: Proc of the 34th IEEE symp on foundations of computer science (FOCS), pp 22–31
7. Awerbuch B, Bartal Y, Fiat A (1998) Distributed paging for general networks. J Algorithms 28(1):67–104. Also appeared in Proc of the 7th SODA, pp 574–583 (1996)
8. Bartal Y (1996) Distributed paging. In: Dagstuhl workshop on online algorithms, pp 97–117
9. Bartal Y (1996) Probabilistic approximations of metric spaces and its algorithmic applications. In: Proc of the 37th IEEE symp on foundations of computer science (FOCS), pp 184–193
10. Bartal Y (1998) On approximating arbitrary metrics by tree metrics. In: Proc of the 30th ACM symp on theory of computing (STOC), pp 161–168
11. Bartal Y, Fiat A, Rabani Y (1995) Competitive algorithms for distributed data management. J Comput Syst Sci 51(3):341–358. Also appeared in Proc of the 24nd STOC, pp 39–50 (1992)
12. Bartal Y, Charikar M, Indyk P (2001) On page migration and other relaxed task systems. Theor Comput Sci 268(1):43–66. Also appeared in Proc of the 8th SODA, pp 43–52 (1997)
13. Ben-David S, Borodin A, Karp RM, Tardos G, Wigderson A (1994) On the power of randomization in online algorithms. Algorithmica 11(1):2–14. Also appeared in Proc of the 22nd STOC, pp 379–386 (1990)
14. Bienkowski M (2005) Dynamic page migration with stochastic requests. In: Proc of the 17th ACM symp on parallelism in algorithms and architectures (SPAA), pp 270–278
15. Bienkowski M, Byrka J (2005) Bucket game with applications to set multicover and dynamic page migration. In: Proc of the 13th European symp on algorithms (ESA), pp 815–826
16. Bienkowski M, Korzeniowski M (2005) Dynamic page migration under Brownian motion. In: Proc of the European conf in parallel processing (Euro-Par), pp 962–971
17. Bienkowski M, Korzeniowski M, Meyer auf der Heide F (2004) Fighting against two adversaries: page migration in dynamic networks. In: Proc of the 16th ACM symp on parallelism in algorithms and architectures (SPAA), pp 64–73
18. Bienkowski M, Dynia M, Korzeniowski M (2005) Improved algorithms for dynamic page migration. In: Proc of the 22nd symp on theoretical aspects of computer science (STACS), pp 365–376
19. Bienkowski M, Byrka J, Korzeniowski M, Meyer auf der Heide F (2009) Optimal algorithms for page migration in dynamic networks. J Discrete Algorithms 7(4):545–569
20. Black DL, Sleator DD (1989) Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University
21. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge
22. Chrobak M, Larmore LL, Reingold N, Westbrook J (1997) Page migration algorithms using work functions. J Algorithms 24(1):124–157. Also appeared in Proc of the 4th ISAAC, pp 406–415 (1993)
23. Fakcharoenphol J, Rao S, Talwar K (2004) A tight bound on approximating arbitrary metrics by tree metrics. J Comput Syst Sci 69(3):485–497. Also appeared in Proc of the 35th STOC, pp 448–455 (2003)
24. Fiat A, Karp RM, Luby M, McGeoch LA, Sleator DD, Young NE (1991) Competitive paging algorithms. J Algorithms 12(4):685–699
25. Fleischer R, Seiden SS (2000) New results for online page replication. In: Proc of the 3rd int workshop on approximation algorithms for combinatorial optimization (APPROX), pp 144–154
26. Fleischer R, Głazek W, Seiden SS (2004) New results for online page replication. Theor Comput Sci 324(2–3):219–251
27. Głazek W (1999) Lower and upper bounds for the problem of page replication in ring networks. In: Proc of the 24th int symp on mathematical foundations of computer science (MFCS), pp 273–283
28. Głazek W (2001) Online algorithms for page replication in rings. Theor Comput Sci 268(1):107–117
29. Imase M, Waxman BM (1991) Dynamic Steiner tree problem. SIAM J Discrete Math 4(3):369–384
30. Karlin AR, Manasse MS, Rudolph L, Sleator DD (1988) Competitive snoopy caching. Algorithmica 3(1):77–119. Also appeared in Proc of the 27th FOCS, pp 244–254 (1986)
31. Karp RM (1992) On-line algorithms versus off-line algorithms: how much is it worth to know the future. In: Proc of the IFIP 12th world computer congress, pp 416–429
32. Koga H (1993) Randomized on-line algorithms for the page replication problem. In: Proc of the 4th int symp on algorithms and computation (ISAAC), pp 436–445
33. Lund C, Reingold N, Westbrook J, Yan DCK (1999) Competitive on-line algorithms for distributed data management. SIAM J Comput 28(3):1086–1111. Also appeared as on-line distributed data management in Proc of the 2nd ESA, pp 202–214 (1994)
34. Matsubayashi A (2008) Uniform page migration on general networks. Int J Pure Appl Math 42(2):161–168

35. McGeoch LA, Sleator DD (1991) A strongly competitive randomized paging algorithm. Algorithmica 6(6):816–825
36. Sleator DD, Tarjan RE (1985) Amortized efficiency of list update and paging rules. Commun ACM 28(2):202–208
37. Vazirani VV (2001) Approximation algorithms. Springer, Berlin
38. Westbrook J (1994) Randomized algorithms for the multiprocessor page migration. SIAM J Comput 23:951–965. Also appeared in Proc of the DIMACS workshop on on-line algorithms, pp 135–149 (1992)
39. Westbrook J, Yan DCK (1995) The performance of greedy algorithms for the on-line Steiner tree and related problems. Theory Comput Syst 28(5):451–468. Also appeared in Proc of the 3rd WADS, pp 622–633 (1993)

Marcin Bienkowski is currently an assistant professor in the Computer Science and Mathematics Department at the University of Wroclaw, Poland. He received his Ph.D. degree in 2005 from the University of Paderborn, Germany, where he worked in the Algorithms and the Complexity Theory group. His research interests include online, approximation, and distributed algorithms.