

A2 : Universal basic services for parallel systems

Marcin Bienkowski, Martin Gairing, Georg Kliewer,
Friedhelm Meyer auf der Heide and Burkhard Monien *

Heinz Nixdorf Institute and Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn,
Germany

Abstract. Basic services like, e.g., routing, load balancing, or data management are essential for efficient usage of parallel and distributed systems. Therefore, the subproject A2 of our CRC376 has focused on design, analysis and implementation of such services. In this report we briefly survey the work of this subproject from 1995 until today. The main part of the report gives more detailed surveys on three of our very recent topics: Dynamic page migration for mobile wireless networks, selfish routing in non-cooperative networks, and exact graph partitioning for process management in parallel systems.

1 Introduction

Within the subproject A2 "Universal Basic Services" of our CRC 376 "Massively Parallel Computing: Algorithms, Design Methods, Applications" we develop services for users of parallel and distributed systems. The goal is to provide a simple and efficient access to the resources like computation time, data storage, memory pages, and communication bandwidth. We have started with investigating tightly coupled massively parallel systems. Over the years, we have more and more extended our focus so that also loosely coupled systems (e.g. Internet) and more dynamic systems like mobile networks are considered. The rest of this introduction gives a very short overview on the major topics we contributed to in this subproject. We restrict the references mainly to PhD-theses, a full list can be found on the Web-page of the CRC.

In the beginning of the SFB data management and load balancing were in the focus of our research. We have developed a distributed data management system which provides transparent and fast access to the shared data objects from any processor in the network. Our first scenario was inspired by early work on PRAM simulation. We have investigated redundant, randomized schemes for static data distribution in networks. This work has resulted into a subproject on Storage Networks which has initiated a so-called Transfer Project where we work towards prototypes and products for storage virtualisation. The PhD-theses of Kay Salzwedel and André Brinkmann resulted from this "transfer" efforts. Our theoretical work has contributed significantly to many aspects of the redundant version of the balls-into-bins-game, which can be seen as the combinatorial basis both for our redundant static data management schemes and for many load balancing approaches. The PhD-theses of Berthold Vöcking, Klaus Schröder, and Petra Berenbrink contribute to this topic.

Our dynamic data management strategies describe how copies of the data objects are distributed and how read and write accesses are served. In order to provide efficient access to shared data objects, the communication overhead, modeled by the congestion, should be as small as possible. We presented online data management strategies for different types of networks and prove optimal or close-to-optimal competitive ratio bounds on the congestion produced by our strategies. Furthermore, we presented data management algorithms that for any given network topology, and any given sequence of requests obtain close to optimum congestion. Our algorithms serve all requests using only local information stored at the network nodes. This allows

* young@rpg.pl, gairing@upb.de, geokl@upb.de, fmadh@upb.de, bm@upb.de

an easy and efficient implementation in a distributed environment, as we have demonstrated in our DIVA library. The PhD-theses of Berthold Vöcking, Matthias Westermann, Harald Räcke contribute to this research topic. A new approach where we incorporate mobility of the network is described in more detail in Section 2. This work is the topic of Marcin Bienkowski's PhD-thesis.

We have also introduced a universal load balancing system which provides numerous load balancing techniques. It supports various programming paradigms and also the combination of different paradigms in the same application. The system is now in use in different application areas. (*Thomas Decker, Reinhard Lüling*)

Routing strategies represent another basic service which use available communication infrastructure and enable the exchange of information between processors. We worked on universal routing strategies, that is, strategies which can be applied to arbitrary network topologies. By introducing the routing number of a network, we offered a rigorous approach to measure the quality of routing protocols. We introduced new universal protocols for both the store-and-forward and the wormhole routing model. For more information see Christian Scheideler's PhD-thesis.

Large-scale networks often lack a central control instance. The distributed entities try to optimize their own goals without regard to the overall performance of the system. We work on various scenarios for selfish routing and scheduling and study the system performance like the price of anarchy in networks with selfish users and the computational complexity of Nash equilibria. (*Martin Gairing, Thomas Lücking, Manuel Rode*)

In several projects of the SFB the following question must be answered: How good is the quality of developed heuristical methods? We are working in the area of optimization (especially linear mixed-integer optimization) and we develop optimal solution methods for problems which arise in the SFB. For the problem of graph partitioning we developed algorithms which compute optimal solutions in a reasonable time. We use the results as an evaluation for heuristical algorithms for graph partitioning which are developed in the project A3. (*Meinolf Sellmann, Norbert Sensen*)

In the next three sections we present three topics considering different aspects of our work in the project A2. In Section 2 we introduce data management strategies in dynamically changing network topologies. In Section 3 we give a short overview on selfish routing in non-cooperative networks and present then an algorithm for computation of a Nash equilibrium for a routing game. In Section 4 we present one application of optimization methods developed for the SFB: the calculation of an exact graph partitioning for process management.

2 Dynamic page migration

The traditional approach of storing the shared data in one or a few central repositories does not scale up well with the increase of the network size, and is therefore inherently inefficient. We investigated *data management strategies* that try to exploit *topological locality*, i.e., try to migrate the shared data in the network between participating nodes in such a way that a node accessing a data item finds it “nearby” in the network. This problem can be modeled as an online problem. This means that the accesses to the shared objects appear online, and the data management algorithm has to decide, without knowledge of the future accesses, whether it is worth to change the positions of objects' copies. While several models have been proposed, we were mainly dealing with the classical, most basic of these data management problems, called *Page Migration*.

The page migration model, introduced in [16], reflects the following assumptions. There is only *one copy of one shared object* (further referred to as the *memory page*) of size D , stored in the local memory of one of the network nodes. The model is *non-uniform*, which means that migrating the whole object is much more expensive than accessing one unit of data from it.

The network is modelled as a connected, undirected graph, where each edge e has an associated cost $c(e)$ of sending one unit of data over the corresponding communication channel. In case of wired networks, this cost might 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 is defined as the sum of costs of edges on the cheapest path between v_a and v_b .

A problem instance is a sequence of nodes $(\sigma_t)_t$, which want to access (read or write) one unit of data from the page. In one step t , one node σ_t issues a request to the node holding the page and appropriate data is sent back. For such a request, a page migration algorithm is charged a cost of sending one unit of data between σ_t and the node holding the page. At the end of each time step the algorithm may *move (migrate)* the page to an arbitrary node. Such a transaction incurs a cost which is D times greater than the cost of sending one unit of data between these two nodes. The goal is to compute a schedule of page movements to minimize the total cost. Moreover, the decisions have to be made online, i.e., in time step t solely on the part of the input up to step t .

In contrast to previous works on data management in networks, we focused on the page migration in a *dynamic* setting. We assumed that the network is no longer static, but is subject to change, and the costs of communication between nodes may change with time. Such a situation is typical in mobile ad-hoc networks, but occurs also in large distributed systems, which are used concurrently by many applications and users. Thus, we have to deal with two sources of online events, namely the requests from nodes to the data item and the changes in the network. Our contribution includes both *modelling* and *algorithm design and analysis*, arising from the combination of data management with the network dynamics.

2.1 Previous work

Previous results on the page migration problem concentrated on the static networks. To measure the performance of online strategies for this problem, the competitive analysis was used. This kind of evaluation, primarily introduced by Sleator and Tarjan [51], essentially compares the cost of an online solution to the cost of the optimal offline strategy. In the following we assume that an optimal solution is denoted by OPT, and for any algorithm ALG, $C_{\text{ALG}}(I)$ denotes the cost of this algorithm on input sequence $I = (\sigma_t)_t$. An online deterministic algorithm ALG is *c-competitive*, if there exists a constant A , such that for any input sequence I , it holds that

$$C_{\text{ALG}}(I) \leq c \cdot C_{\text{OPT}}(I) + A .$$

For a randomized algorithm ALG we replace its cost in the definition above by its expected value $\mathbf{E}[C_{\text{ALG}}(I)]$, where expectation is computed over all random choices made by ALG.

First randomized solutions presented by Westbrook [54] were a memoryless algorithm, which was 3-competitive against an adaptive adversary, and a phase-based algorithm whose competitive ratio against an oblivious adversary tends to 2.618 as D goes to infinity. The former result was proven to be tight in [7, 5]. On the other hand, the exact competitive ratio against an oblivious adversary is not a completely settled issue. Currently, the best known lower bound, $2 + \frac{1}{2D}$, is due to Chrobak et al. [18]. It is matched only for certain topologies, like trees or uniform networks (see [18] and [36], respectively).

The currently best deterministic strategy is a 4,086-competitive, phase-based algorithm MOVE-TO-LOCAL-MIN presented in [6]. On the other hand, [18] showed a network with a lower bound of approximately 3.148.

There exist also many extensions of page migration that allow more flexible data management in networks. One of the possible generalizations is allowing more than one copy of an object to exist in the network. A basic version of this extension, where only one shared object is present in the system, is called *file allocation*, and was examined in [1, 7, 36]. In particular, the results include $O(\log n)$ -competitive strategies for this problem.

If multiple objects are present in the network and the nodes' memory capacities are limited, then simply running a file allocation scheme independently for each single object is no longer feasible. This *distributed paging* problem was considered in [7, 2, 3]. In particular, Awerbuch et al. [3] gave the deterministic $O(\text{polylog } n)$ -competitive algorithm, under the assumption that the online algorithm has $O(\text{polylog } n)$ times more memory than the optimal algorithm.

2.2 New model for network dynamics

In the past, an application executed on a parallel machine was running in a virtually static and invariable environment and one could safely assume that the interconnecting network is predictable and reliable. Such assumptions, which substantially reduced the complexity of the basic services design, ceased to hold when applications started to run in open and unknown networks. In reality the network is usually subject to small continuous changes, like changes in bandwidth capacity, or the changes in the topology induced by node mobility. These network alterations induce the changes in the costs of communication between pairs of nodes.

Basic services for mobile wireless networks and dynamically changing wired networks are a relatively new area. The *topology control* (the problem of computing and maintaining a connected topology stretched on the network nodes) and *routing* in such networks received attention in a past few years [43, 4, 47].

In comparison, basic services related to data management problems in dynamic networks are still in their infancy. Till recently, no theoretical analysis or even experimental evaluation was present in this area. We strived for creating a model, where the costs of point-to-point communication can be changed arbitrarily, as long as the pace of these changes is restricted by, say, an additive constant per step. While this model allows for rigorous analysis, it also covers quite a lot of common cases in reality. This is for example motivated by a reality-close *pedestrian model* by Schindelbauer et al. [48], where mobile stations might be moved arbitrarily by an adversary, as long their speed is bounded.

We formally defined the model in [14] making the following assumptions. The network is modelled as a set of n mobile nodes (processors) labelled v_1, v_2, \dots, v_n . These nodes are placed in a metric space (\mathcal{X}, d) , where the distance between any pair of points from \mathcal{X} is given by the metric d . We assume that the position of each node is a function of time step t . The only restriction we impose is that all the nodes move with a *bounded speed*, i.e., for any node, the distance between its positions in any two consecutive time steps is at most a constant.

As a natural consequence, for any two nodes v_a and v_b , their distance may also change with time; the distance in step t we denote by $d_t(v_a, v_b)$. A tuple describing the positions of all the nodes in time step t is called *configuration in step t* , and is denoted by C_t . Any two nodes are able to communicate directly with each other. 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.

Similarly to the casual page migration, in dynamic networks in each step an algorithm has to serve the request, and then decide, whether it wants to migrate the page to some other node. Precisely, for any algorithm ALG the following stages happen in time step t .

1. The positions of the nodes in the current step are defined by C_t .
2. A node σ_t wants to access one single unit of data from the page. It sends a write or a read request to v_{ALG} , the node holding ALG's page in the current step.
3. ALG serves this request, i.e., it sends a confirmation in case of write, or a requested unit of data in case of read. This transaction incurs a cost $c_t(v_{\text{ALG}}, \sigma_t)$.
4. ALG optionally moves the page to another node of its choice, called a *jump candidate*. A movement to v_{ALG}^* incurs a cost $D \cdot c_t(v_{\text{ALG}}, v_{\text{ALG}}^*)$.

We further refer to this problem as *Dynamic Page Migration* (DPM). The goal is, again, to construct a schedule of page movements to minimize the total cost of communication for any input, i.e., for any pair of sequences $(C_t)_t, (\sigma_t)_t$.

2.3 Our contribution

Like in the page migration case, the problem of minimizing the total cost incurred is relatively easy, if both $(C_t)_t$ and $(\sigma_t)_t$ are given in *offline* setting, i.e., if an algorithm may read the whole input beforehand. In [10] we constructed a dynamic programming approach, which is able to find an optimal schedule of page movements for any instance of the DPM problem consisting of T steps, using $O(T \cdot n^2)$ operations and $O(T \cdot n)$ additional space.

However, as mentioned earlier, DPM has to be primarily solved in an *online* setting, where an algorithm must make its decisions (where to move the page) in time step t , exclusively on the sequence $C_0, C_1, \sigma_1, C_2, \sigma_2, \dots, C_t, \sigma_t$. To evaluate an online strategy for the DPM problem, we use competitive analysis. Since the input sequence consists of two practically independent streams, $(\sigma_t)_t$ describing the request patterns and $(C_t)_t$ reflecting the changes in network topology, it is reasonable to assume that they are created by two separate adversarial entities, a *request adversary* and a *configuration (network) adversary*.

We designed algorithms for different powers of adversaries, and rigorously analyze them using competitive analysis and its variants. However, we have not only to precise the power of a single adversary, but also to decide whether these adversaries cooperate in creating an input sequence. This yields different scenarios depending on the ways in which these adversaries interact.

(a) *Adversarial (cooperative) scenario*. The most straightforward modelling occurs when both adversaries may cooperate to create the combined input sequence. In fact, this is equivalent to having one adversary capable of constructing the whole input sequence.

We constructed a deterministic strategy, which is $O(\min\{n \cdot \sqrt{D}, D\})$ -competitive. Its main non-trivial building block, algorithm MARK, presented in [12], achieves a competitive ratio of $O(n \cdot \sqrt{D})$. All algorithms for the page migration in static networks used one main paradigm: moving the page to a so called *gravity center*, which was the node lying “in the middle” of the last few requests. We proved that in the dynamic setting such an approach can no longer guarantee a non-trivial competitive ratio. Instead, an algorithm has to consider not only gravity centers, but also nodes lying close to gravity centers as its potential jump candidates. This is exactly what MARK algorithm does. It works in phases consisting of a fixed number of steps. At the end of each phase, it marks nodes which were far away from the recent requests, and

moves to arbitrarily chosen unmarked node. The latter nodes are provably close to the gravity center.

We also show that if an algorithm may use random bits, then it can perform asymptotically as well as MARK, even without having any state information. The $O(n \cdot \sqrt{D})$ -competitive algorithm DIST presented in [14] can also be extended to an $O(\min\{n \cdot \sqrt{D}, D\})$ -competitive randomized strategy, which is competitive even if the adversary is adaptive, i.e., it may see the random bits used by the algorithm. Due to the lower bound shown in [14], these deterministic and randomized strategies are up to a constant factor optimal.

Furthermore, in [11] we were able to randomize algorithm MARK, so that the resulting algorithm chooses jump candidates randomly amongst not yet marked nodes, according to their distance to the gravity center. By choosing an appropriate probability distribution, we reduced the factor n appearing in the competitive ratio to $\sqrt{\log n}$. The resulting algorithm EBM is therefore $O(\sqrt{D} \cdot \log n, D)$ competitive against an oblivious adversary (the one which cannot see the algorithm's random bits). In [12], we showed an almost matching lower bound $\Omega(\min\{\sqrt{D} \cdot \log n, D^{2/3}\})$.

All the presented competitive ratios are strict, which means that the constant A occurring in the definition of the competitive ratio is equal to zero. However, the competitive ratios of the best possible algorithms for DPM problem are large, even against the weakest, oblivious adversaries. It can be inferred that the poor performance of algorithms for this scenario is caused by the fact that the network and request adversaries might combine and synchronize their efforts in order to destroy our algorithm. Such a cooperation is unfair and rather unrealistic, it is however unclear how to forbid it. Therefore we propose that the DPM problem could be analyzed in another extreme case, where one of the adversaries is replaced by a stochastic process. This leads to another two scenarios.

(b) *Brownian motion scenario.* In this scenario, which we defined in [14], requests are given by the adversary, but the movement of nodes is random. Precisely, the mobile nodes perform a random walk on a bounded area of diameter B , and the request adversary dictates which nodes issue requests during runtime. By *area* we mean a d -dimensional discrete torus or mesh of diameter B , where d is a constant.

The request adversary is “oblivious”, i.e., it has to create the whole request sequence $(\sigma_t)_t$ in advance, without knowledge of the actual configuration sequence $(C_t)_t$ induced by a random walk. The definition of competitiveness has to be adapted appropriately to reflect the fact that the input sequence is created both by an adversary and a stochastic process. Motivated by the research in the smoothed analysis of online algorithms [8, 46] we introduced the following notion. A deterministic algorithm ALG is c -competitive with probability p , if there exists a constant A , such that for all request sequences $(\sigma_t)_t$ it holds that

$$\Pr_{(C_t)_t} \left[C_{\text{ALG}}((C_t)_t, (\sigma_t)_t) \leq c \cdot C_{\text{OPT}}((C_t)_t, (\sigma_t)_t) + A \right] \geq p ,$$

where the probability is taken over all possible configuration sequences generated by the random movement.

In [14, 13] we presented a simple *majority* approach MAJ, which after a phase consisting of some fixed number of steps, moves to the node which issued a majority of the requests in this phase. It appears that MAJ is $O(\min\{\sqrt[4]{D}, n\} \cdot \text{polylog}(B, D, n))$ -competitive in 1-dimensional areas. The result can be extended to any constant-dimensional areas for $B \geq \sqrt{D}$. The ratio is achieved with high probability, which means that the probability p occurring in the definition above can be amplified to $1 - (B \cdot D)^{-\gamma}$, for sufficiently long input sequences.

(c) *Stochastic requests scenario.* This scenario, discussed in [9], is symmetric to the Brownian motion one. It assumes that requests appear with given frequencies, i.e., in step t , σ_t is a node chosen randomly according to a fixed probability distribution π . We took a performance measure identical to the one used in the Brownian motion case, but with the roles of configuration and request sequences swapped.

We presented an algorithm MOVE-TO-FIRST-REQUEST, achieving strict $O(1)$ -competitive ratio, with high probability. This algorithm migrates the page each ℓ steps to the node which has just issued the request, where ℓ is a carefully chosen constant. Moreover, without hindering the competitive ratio, the algorithm can be slightly modified to also handle the case where the communication cost between two nodes is proportional to the distance between them to some fixed power α . For the case of wireless radio networks, one can choose the parameter α to respect a *propagation exponent* of the medium (see, e.g., [44]). For example, by setting $\alpha = 2$, the cost definition reflects the energy consumption used to send a message in the ideally free space along a given distance. Thus, this result minimizes, up to a constant factor, the total energy used in the system.

3 Selfish Routing in Non-Cooperative Networks

3.1 Framework

Large-scale traffic and communication networks, like e.g. the internet, telephone networks, or road traffic systems often lack a central regulation for several reasons: The size of the network may be too large, the network may be dynamically evolving over time, or the users of the network may be free to act according to their private interest, without regard to the overall performance of the system. Besides the lack of central regulation even cooperation of the users among themselves may be impossible due to the fact that the users may not even know each other. Networks with non-cooperative users have already been studied in the early 1950s in the context of road traffic systems (see e.g. [53]) Recently, motivated by non-cooperative systems like the internet, combining ideas from game theory and theoretical computer science has become increasingly important (see e.g. [41]).

An environment, which lacks a central control unit due to its size or operational mode, can be modeled as a non-cooperative game. Users selfishly choose their private strategies, which in our environment correspond to paths (or probability distributions over the paths) from their *sources* to their *destinations*. When routing their *traffics* according to the strategies chosen, the users will experience an *expected latency* caused by the traffics of all users sharing edges. Each user tries to minimize its *private cost*, expressed in terms of its expected latency. This often contradicts the goal of optimizing the *social cost* which measures the global performance of the whole network. Such networks are called *non-cooperative networks*. The degradation of the global performance due to the selfish behavior of its users is often termed *price of anarchy* [41, 45] and measured in terms of the *coordination ratio*. The theory of Nash equilibria [39, 40] provides us with an important tool for environments of this kind: A *Nash equilibrium* is a state of the system such that no user can decrease its private cost by unilaterally changing its strategy.

The concept of Nash equilibrium [39, 40] has become an important mathematical tool for analyzing the behavior of selfish users in non-cooperative systems. It has been shown by Nash that a Nash equilibrium exists under fairly broad circumstances. Many algorithms have been developed to compute a Nash equilibrium in a general game (see [37] for an overview). Although the celebrated result of Nash [39, 40] guarantees the existence of a Nash equilibrium for any finite strategic game, the complexity of computing a Nash equilibrium in general games is wide

open even if only two users are involved. This problem has been advocated as one of the most important open problems in theoretical computer science today [41].

3.2 Contribution

In recent years we extensively studied various models of selfish routing and scheduling. One task that was successfully hit had to do with the computation of an optimum pure Nash equilibrium for a selfish routing game on parallel links. Roughly speaking, an optimal (pure) Nash equilibrium minimizes a certain measure of social performance of the system. This social performance is usually coined as social cost. We pioneered the idea of Nashification algorithms; these are algorithms converting any arbitrary assignment into a Nash equilibrium without increasing social cost. This simple idea has been the driving force towards developing some Nashification algorithms for some particular instances of the selfish routing game [20, 25, 28]. Today, Nashification is widely considered an algorithmic paradigm for the design and analysis of algorithms computing (or approximating) a pure Nash equilibrium.

In another line of research we focused on the study of *worst case Nash equilibria*. In this setting we introduced the fully mixed Nash equilibrium conjecture. Roughly speaking this conjecture states that the fully mixed Nash equilibrium (which is some particular type of Nash equilibria) is the worst possible Nash equilibrium with respect to some social cost. This conjecture was explicitly formulated in [28] and further studied in [35]. For social cost defined as the sum of individual costs the conjecture holds [27, 34] but it was disproved for social cost defined as the expected maximum congestion [23].

In a different front we intensively work towards a characterization, and an improved understanding of properties, of various models for selfish routing and scheduling [19–21, 25–30, 34, 35]. The characterization has been based on the definitions of individual cost and social cost. A significant milestone has been the recent model of Bayesian routing [30].

3.3 A Nashification Algorithm for Restricted Parallel Links

As one of the highlights from our work, we now present a Nashification algorithm for a routing game on restricted parallel links. Here links are identical, however a user is only allowed to route its traffic along a link from a subset of allowed links for the user. Our algorithm is called NASHIFY-RESTRICTED and was first presented in [25]. It identifies some natural connections between the problem of computing a Nash equilibrium and *network flow* problems.

In order to present the algorithm, we first introduce some notations and we show how to represent a (partial) pure assignment via a residual network. We then introduce a blocking flow algorithm, called UNSPLITTABLE-BLOCKING-FLOW, which is the key routine of our Nashification algorithm. Finally, we describe how UNSPLITTABLE-BLOCKING-FLOW can be used to alter a pure assignment such that the social cost does not increase and the resulting assignment is a Nash equilibrium.

Notation. We consider a network consisting of a set $M = \{1, 2, \dots, m\}$ of m parallel links from a *source* node to a *sink* node. Each *users* from a set $U = \{1, 2, \dots, n\}$ of n users wishes to route a particular amount of traffic along a (non-fixed) link from source to sink. Denote w_i the (integer) *traffic* of user $i \in U$. Assume, without loss of generality, that $w_1 \geq \dots \geq w_n$, and denote $W = \sum_{i \in U} w_i$ (the *total traffic*). The *traffic vector* \mathbf{w} is the tuple of all user traffics.

A *strategy* for a user $i \in U$ is some specific link. The key distinguishing feature of the restricted parallel links model is that there is associated with each user $i \in U$ a *strategy set* $A_i \subseteq M$, as the set of *allowed links* for user i ; thus, user i can only be assigned to a link from A_i .

Denote $A = \sum_{i \in U} |A_i|$ the total size of strategy sets. An *assignment* $\mathbf{L} = \langle l_1, \dots, l_n \rangle$ is a tuple of strategies, one for each user.

For the assignment \mathbf{L} , the *load* δ_j on link j is the sum of traffics of all users assigned to link j ; thus, $\delta_j = \sum_{k: l_k=j} w_k$. The *Individual Cost* λ_i of user $i \in U$ in assignment \mathbf{L} is the *latency* of the link it chooses; that is, $\lambda_i = \delta_{l_i}$.

Associated with a traffic vector \mathbf{w} and an assignment \mathbf{L} is the *Social Cost* denoted $SC(\mathbf{w}, \mathbf{L})$, which is the maximum, over all links, latency due to the load through the link; thus,

$$SC(\mathbf{w}, \mathbf{L}) = \max_{j \in M} \sum_{k: l_k=j} w_k.$$

The *Optimum* associated with a traffic vector \mathbf{w} is the least possible, over all assignments, of the maximum, over all links, latency due to the load through the link; thus,

$$OPT(\mathbf{w}) = \min_{\mathbf{L} \in M^n} SC(\mathbf{w}, \mathbf{L}).$$

We are interested in a special class of assignments called Nash equilibria [39, 40] that we describe below. Say that a user $i \in U$ is *satisfied* in assignment \mathbf{L} if for all links $j \in A_i$,

$$\lambda_i \leq \frac{\delta_j + w_i}{c_i};$$

thus, a satisfied user has no incentive to unilaterally change its strategy. An *unsatisfied* user is one that is not satisfied. The assignment \mathbf{L} is a *Nash equilibrium* if all users are satisfied.

In the following, we present a (partial) pure assignment with help of a residual network.

Definition 1. Given a (partial) pure assignment $\mathbf{L} = \langle l_1, \dots, l_n \rangle$, we define a directed bipartite graph $G_{\mathbf{L}} = (V, E_{\mathbf{L}})$, where $V = M \cup U$ such that each link is represented by a node in M and each user is represented by a node in U . Furthermore, $E_{\mathbf{L}} = E_{\mathbf{L}}^1 \cup E_{\mathbf{L}}^2$ with

$$E_{\mathbf{L}}^1 = \{(j, i) \mid j \in M, i \in U, j = l_i\}, \text{ and}$$

$$E_{\mathbf{L}}^2 = \{(i, j) \mid i \in U, j \in M, j \in A_i \setminus \{l_i\}\}.$$

For a total pure assignment \mathbf{L} , we use the graph $G_{\mathbf{L}}$ from Definition 1 to define a graph $G_{\mathbf{L}}(w)$ where V stays the same, but from $E_{\mathbf{L}}$ we now only consider edges

$$E_{\mathbf{L}}(w) = E_{\mathbf{L}} \setminus \{(i, j) \mid i \in U, j \in M, w_i > w\}.$$

This means that all users $i \in U$ with $w_i > w$ stay assigned to their link. We use $G_{\mathbf{L}}$ instead of $G_{\mathbf{L}}(w)$ if it is clear from the context which w is used.

Unsplittable-Blocking-Flow. We now introduce an algorithm, called UNSPLITTABLE-BLOCKING-FLOW. Starting with any integer $w \in \mathbb{N}$ and any pure assignment \mathbf{L} , we use an integer a to control the approximation of an optimum assignment. The intention is to find an a which is a lower bound on $OPT(\mathbf{w})$, and then to compute a pure assignment \mathbf{L}' with $SC(\mathbf{w}, \mathbf{L}') \leq a + w$. For any integer a , we partition the set of links M into three subsets:

$$M^- = \{j \in M \mid \delta_j(\mathbf{L}) \leq a\}$$

$$M^0 = \{j \in M \mid a + 1 \leq \delta_j(\mathbf{L}) \leq a + w\}$$

$$M^+ = \{j \in M \mid \delta_j(\mathbf{L}) \geq a + w + 1\}$$

In this setting, we do not have a dedicated source or sink. However, at each time nodes in M^+ and M^- can be interpreted as source and sink nodes, respectively. Note, that those sets change over time.

UNSPLITTABLE-BLOCKING-FLOW combines ideas from blocking flows with the idea of pushing users without splitting them. Roughly speaking, algorithm UNSPLITTABLE-BLOCKING-FLOW shifts users so that the latencies of links from M^- are never decreased, the latencies of links from M^+ are never increased, and links from M^0 remain in M^0 . The algorithm is controlled by a *height function* $h : V \rightarrow \mathbb{N}_0$ with $h(j) = \text{dist}_{G_{\mathbf{L}}(w)}(j, M^-)$ for all $j \in V$. We call an edge (u, v) *admissible* if $h(u) = h(v) + 1$. In an *admissible path*, all edges are admissible. For each node $u \in V$ with $0 < h(u) < \infty$, define $S(u)$ to be the *set of successors* of node u ; this is the set of nodes to which u has an admissible edge, so that

$$S(u) = \{v \in V \mid (u, v) \in E_{\mathbf{L}} \text{ and } h(u) = h(v) + 1\}.$$

Note that $S(u)$ also defines the set of admissible edges leaving u . Let $s(u)$ be the first node in a list implementation of the set $S(u)$. We proceed to define:

Definition 2. A link $j \in M$ with $0 < h(j) < \infty$ is called *helpful* if $\delta_j(\mathbf{L}) \geq a + 1 + w_{s(j)}$.

Lemma 1. Let v_0 be a helpful link of minimum height. Then, there exists a sequence v_0, \dots, v_r , where $v_{2i} \in M$ for all $0 \leq i \leq r/2$ and $v_{2i+1} \in U$ for all $0 \leq i < r/2$ such that

- (1.) $(v_i, v_{i+1}) \in E_{\mathbf{L}}$ and $h(v_i) = h(v_{i+1}) + 1$,
- (2.) $\delta_{v_0}(\mathbf{L}) \geq a + 1 + w_{s(v_0)}$,
- (3.) $a + 1 \leq \delta_{v_{2i}}(\mathbf{L}) + w_{s(v_{2i-2})} - w_{s(v_{2i})} \leq a + w$ for all $0 < i < r/2$,
- (4.) $\delta_{v_r}(\mathbf{L}) + w_{s(v_{r-2})} \leq a + w$.

We are now ready to present the algorithm UNSPLITTABLE-BLOCKING-FLOW, stated as Algorithm 1. Initially, the height function h is computed as the distance in $G_{\mathbf{L}}(w)$ of each node to the set M^- of nodes. Then, the algorithm proceeds in phases. In each phase, first the minimum height $d = h(v)$ of a node $v \in M^+$ is computed. Inside each phase, we do not update the height function, but we successively choose a helpful link v of minimum height and we push users along the helpful path induced by v and adjust the pure assignment accordingly. In order to update $G_{\mathbf{L}}(w)$, we have to change the direction of two arcs for each user push. The phase ends when there exists no further admissible path from a node $v \in M^+$ with $h(v) = d$ to some node in M^- . Before the new phase starts, we recompute h and we check whether we need to start a new phase or not. UNSPLITTABLE-BLOCKING-FLOW stops when either $M^- = \emptyset$ or for all $v \in M^+$ we have $h(v) = \infty$.

We showed [25] that UNSPLITTABLE-BLOCKING-FLOW decreases the maximum load and increases the minimum load on the links, respectively (Lemma 2). Moreover, we showed properties of the resulting pure assignment \mathbf{L}' (Lemma 3), and that UNSPLITTABLE-BLOCKING-FLOW can be implemented to run in $O(mA)$ time (Theorem 1).

Lemma 2. For the pure assignment \mathbf{L}' computed by UNSPLITTABLE-BLOCKING-FLOW(\mathbf{L}, a, w), we have

$$\begin{aligned} \max_{j \in M} \delta_j(\mathbf{L}') &\leq \max_{j \in M} \delta_j(\mathbf{L}), \quad \text{and} \\ \min_{j \in M} \delta_j(\mathbf{L}') &\geq \min_{j \in M} \delta_j(\mathbf{L}). \end{aligned}$$

Algorithm 1 (UNSPLITTABLE-BLOCKING-FLOW)

Input: a pure assignment \mathbf{L} and positive integers a, w

Output: a pure assignment \mathbf{L}'

```
(1)  begin
(2)  compute  $h$ ;
(3)   $\mathbf{L}' \leftarrow \mathbf{L}$ ;
(4)  while  $M^- \neq \emptyset$  and there exists a  $v \in M^+$  with  $h(v) < \infty$  do
(5)     $d \leftarrow \min_{v \in M^+} (h(v))$ ;
(6)    while there exists an admissible path from  $v \in M^+, h(v) = d$ , to  $M^-$  do
(7)      choose helpful link  $v$  of minimum height;
(8)      push users along helpful path defined by  $v$ ;
(9)      update  $\mathbf{L}', G_{\mathbf{L}'}(w)$ ;
(10)   recompute  $h$ ;
(11) return  $\mathbf{L}'$ ;
(12) end
```

Lemma 3. *For the pure assignment \mathbf{L}' computed by UNSPLITTABLE-BLOCKING-FLOW(\mathbf{L}, a, w), one of the following conditions holds:*

- (1.) $M^-(\mathbf{L}') = \emptyset$.
- (2.) $M^+(\mathbf{L}') = \emptyset$.
- (3.) *There exists some set of links $B \subset M$ such that*
 - (a.) $\delta_j(\mathbf{L}') \geq a + 1$ for all $j \in B$, and
 - (b.) $\delta_j(\mathbf{L}') \leq a + w$ for all $j \in M \setminus B$, and
 - (c.) $\ell_i \in B$ implies $A_i \subseteq B$ for all $i \in U$ with $w_i \leq w$.

Theorem 1. UNSPLITTABLE-BLOCKING-FLOW can be implemented to run in $O(mA)$ time.

Nashify-Restricted. We now briefly describe how UNSPLITTABLE-BLOCKING-FLOW can be used to convert any pure assignment into a pure Nash equilibrium with non-increased social cost. Our Nashification algorithm first finds an assignment satisfying all users with traffic w_1 by recursively applying UNSPLITTABLE-BLOCKING-FLOW. In the recursive procedure we make extensive use of Lemma 3.

We then fix the assignment of all users with traffic w_1 and proceed with the next smaller traffic while making sure that all fixed users stay satisfied. To make sure that all fixed users stay satisfied, we introduce lower and upper bounds on the load of the links, such that the load of each link is always in its bounds, the lower bound only increases and the upper bound only decreases. This is done until all users are satisfied. In order to achieve this, we again make extensive use of algorithm UNSPLITTABLE-BLOCKING-FLOW. The total running time of NASHIFY-RESTRICTED is $O(nmA(\log W + m^2))$. For a detailed description of our Nashification algorithm we refer to [25].

4 Exact Graph Partitioning

4.1 Motivation and contribution

Graph Partitioning is the problem of partitioning a set of vertices of a graph into disjoint subsets of a given maximal size such that the number of edges with end points in different subsets is minimized. Graph Partitioning is a very common problem and has a large number of applications. For example circuit layout, compiler design, and load balancing are typical applications

in which Graph Partitioning problems appear. Unfortunately, the Graph Partitioning problem is a NP-hard problem. So in the last years a lot of effort has been spent in the development of fast and good heuristics for the problem, a recent survey is given in [24]. These heuristics often can handle rather large graphs with more than a million vertices and deliver good solutions. In contrast to the development of heuristics only a little expense has been done in the development of exact algorithms. From the NP-hardness fact it is clear that generally only relatively small graphs can be solved exactly. Nevertheless, exact solutions are of interest for applications and for the validation of heuristics.

In the last years, a number of exact approaches for solving the graph partitioning problem have been presented. One approach is presented in [32] by S.E. Karisch, F. Rendl, and J. Clausen who use semidefinite programming relaxations for computing bounds on the objective. In [22], Ferreira et al. present a branch-and-cut algorithm for the problem. The same approach is followed by L. Brunetta, M. Conforti, and G. Rinaldi in [17]. In [31], E. Johnson, A. Mehrotra, and G. Nemhauser elaborate on a column generation approach for the graph partitioning problem.

In [50], we presented a new lower bound for the bisection width of a graph. It can be obtained by solving a multicommodity flow problem with variable sender volumes. The problem may be viewed as a generalized maximum multicommodity flow problem: we need to compute a maximum multicommodity flow on an undirected network where the commodities have exactly one source, but may have many sinks. Building up on existing techniques for the solution of multicommodity flow problems, we develop in [49] a fully polynomial time approximation scheme (FPTAS). The routine is embedded in a branch&bound-framework for exact graph bisection and experimentally compared with a Lagrangian relaxation based cost-decomposition approach and a barrier LP-solver on various test instances. On top of that, we compare the multicommodity bound and the different algorithms for its computation/approximation with the semidefinite bounding routine developed in [32]. Finally, we prove the performance of our exact graph bisection algorithm by computing the previously unknown bisection width of some DeBruijn- and Shuffle-Exchange-Graphs.

4.2 Bounds on Graph Bisection

Our work bases on the lower bounds on the graph partitioning problem introduced in [50]. In the following, we give a small survey on the main ideas of these bounds:

A well known lower bound (see [33]) on the bisection width can be achieved by embedding a clique with the same number of nodes n into the given graph G . When embedding the clique, an edge between two nodes translates into a path in G . The number of paths that use the same edge e is called the *congestion* of e . The edge with the maximum congestion then determines the congestion of the embedding. If the clique can be embedded with congestion C , we know that G has a bisection width of at least $\frac{n^2}{4C}$. Note that the clique embedding can also be viewed as an integer multicommodity flow problem where every vertex sends a commodity of size one to every other vertex.

First, we note that actually we do not need to enforce the integrality constraints on the flows and thereby can strengthen the bound computed¹. We can further improve on this bound by taking two steps of generalization: First, it can be observed that every single-source multicommodity flow instance (with arbitrary demands and destinations) can be used to compute a lower bound on the bisection width. The critical point is the *CutFlow*, i.e. the amount of flow which

¹ For flows, the congestion of an edge is the amount of flow that is routed through it.

can be ensured to cross every possible bisection of the graph. It is easy to show that $\frac{CutFlow}{C}$ is always a valid lower bound on the bisection problem.

Second, we do not have to select an appropriate multicommodity flow instance by ourselves: Typically, linear programming techniques are used to solve multicommodity flow problems. In [50], we introduced the idea to leave the selection of an appropriate multicommodity flow instance to the linear program by adding some variables and constraints. Two different possibilities with a different degree of freedom for the selection of the multicommodity flow instance have been introduced: the *VarMC* and the *MVarMC* formulations. In the *MVarMC* formulation, every node has the freedom to send a commodity of arbitrary size to each other node. Whereas in the *VarMC* formulation, every node has to send a commodity of arbitrary size to every other node, whereby each destination gets the same share. Experiments have shown that, for the graph bisection problem, the *VarMC* formulation gives equally good bounds when compared with the *MVarMC* formulation. Therefore, in this section, we compare different solution techniques for the computation of the *VarMC*-bound.

4.3 A Branch&Bound Algorithm

Our main goal is the computation of exact solutions for graph bisection problems. Thus, we construct a branch&bound algorithm using the described *VarMC*-bound as lower bound for the problems. A detailed description of the implementation can be found in [50]. In the following, we give a brief survey on the main ideas:

First, we heuristically compute a graph bisection using *PARTY* [42]. Since this start-solution is optimal in most cases, we only have to prove optimality. We use a pure depth first search tree traversal for this purpose. The branching is done on the decision whether two specific vertices $\{v, w\}$ stay in the same partition (join) or if they are separated (split). A join is performed by merging these two vertices into one vertex. A split is performed by introducing an additional commodity from vertex v to vertex w whose entire amount is known to cross the cut. Thus, it can be added to the *CutFlow* completely. The selection of the pair $\{v, w\}$ for the next branching is done with the help of an upper bound on the lower bound. Additionally to this idea, the selection is restricted to pairs $\{v, w\}$ (if any) where one node (say, v) has been split with some other node $u \neq w$ before. Then, split of $\{v, w\}$ implies a join of $\{u, w\}$, of course.

4.4 Optimal Partitioning

To show the behavior on different kinds of graphs, we use four different sets of 20 randomly generated graphs: The set *RandPlan* contains random maximal planar graphs with 100 vertices; the graphs are generated using the same principle as it is used in LEDA [38]. Benchmark set *RandReg* consists of random regular graphs with 100 vertices and degree four; for its generation, the algorithm from Steger and Wormald [52] is used. The set *Random* contains graphs with 44 vertices where every pair $\{v, w\}$ is adjacent with probability 0.2. The set *RandW* consists of complete graphs with 24 vertices where every edge has a random weight in the interval $\{0, \dots, 99\}$.

In most works on exact graph partitioning (see e.g. [17, 32]), sets like *Random* and *RandW* are used for the experiments. We added the sets of random regular and random planar graphs here, because we believe that more structured graph classes should also be considered with respect to their bigger relevance for practical applications.

We observe that with respect to the memory consumption, approximation and cost-decomposition are preferable to the barrier algorithm: When the graphs we consider become larger, for

DeBruijn 9 or Shuffle-Exchange 9, for example, 2 GB main memory are not enough to allow the application of CPLEX. Thus, it cannot be applied to compute the corresponding bisection widths, whereas both cost-decomposition and approximation allowed us to prove a bisection width of 92 and 48, respectively. Apart from being more memory efficient, we observe that cost-decomposition does not allow us to solve our generalized maximum multicommodity flow problem faster than standard LP methods.

We developed an algorithm for the approximation of the VarMC-bound on the bisection width of a graph. It is based on an approximation scheme for maximum multicommodity flows and yields an ε -approximation in time $O^*(m^2/\varepsilon^2)$. We experimented with different practical improvements that have been suggested for this kind of multicommodity flow approximation scheme and were able to improve the practical behavior by using the new enhanced scaling technique.

When comparing the approximation algorithms with a barrier LP-solver and a Lagrangian relaxation based cost-decomposition algorithm, we found that it is favorable to use the approximation scheme, both with respect to the memory consumption and running time. The VarMC-approximation scheme allowed us to compute the bisection width of large graphs, such as DeBruijn 9 (the bisection width is 92), Shuffle-Exchange 9 (48), and Shuffle-Exchange 10 (82), which were unknown and out of the reach of exact graph bisection algorithms before.

5 Acknowledgements

Following persons worked on different topics in the scope of the project A2: Petra Berenbrink, Marcin Bienkowski, André Brinkmann, Thomas Decker, Ralf Diekmann, Torsten Fahle, Rainer Feldmann, Martin Gairing, Georg Kliewer, Miroslaw Korzeniowski, Christof Krick, Thomas Lücking Reinhard Lüling, Friedhelm Meyer auf der Heide, Burkhard Monien, Robert Preis, Harald Räcke, Manuel Rode, Kay Salzwedel, Klaus Schöder, Stefan Schamberger, Christian Scheideler, Meinolf Sellmann, Norbert Sensen, Christian Sohler, Berthold Vöcking, Matthias Westermann.

6 PhD theses in this project (chronological list)

Hier gehoeren noch die Arbeiten von Kay salzwedel, Andre brinkmann, matthias Westermann, manuel Rode .. hin

Christian Scheideler, "Universal routing strategies", PhD thesis, University of Paderborn, 1996.

Reinhard Lüling, "Lastverteilungsverfahren zur effizienten Nutzung paralleler Systeme", PhD thesis, University of Paderborn, 1997.

Berthold Vöcking, "Static and dynamic data management in networks", PhD thesis, University of Paderborn, 1998.

Thomas Decker, "Ein universelles Lastverteilungssystem und seine Anwendung bei der Isolierung reeller Nullstellen", PhD thesis, University of Paderborn, 2000.

Petra Berenbrink, "Randomized allocation of independent tasks", PhD thesis, University of Paderborn, 2000.

Klaus Schröder, "Balls into bins : a paradigm for job allocation, data distribution processes, and routing", PhD thesis, University of Paderborn, 2001.

Meinolf Sellmann, "Reduction techniques in constraint programming and combinatorial optimization", PhD thesis, University of Paderborn, 2002.

Harald Räcke, "Data management and routing in general networks", PhD thesis, University of Paderborn, 2003.

Norbert Sensen, "Lower bounds and exact algorithms for the graph partitioning problem using multicommodity flows", PhD thesis, University of Paderborn, 2003.

Thomas Lücking, "Analyzing models for scheduling and routing", PhD thesis, University of Paderborn, 2005.

References

1. Baruch Awerbuch, Yair Bartal, and Amos Fiat. Competitive distributed file allocation. In *Proc. of the 25th ACM Symp. on Theory of Computing (STOC)*, pages 164–173, 1993.
2. Baruch Awerbuch, Yair Bartal, and Amos Fiat. Heat & dump: Competitive distributed paging. In *Proc. of the 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 22–31, 1993.
3. Baruch Awerbuch, Yair Bartal, and Amos Fiat. Distributed paging for general networks. *Journal of Algorithms*, 28(1):67–104, 1998. Also appeared in *Proc. of the 7th SODA*, pages 574–583, 1996.
4. Baruch Awerbuch, André Brinkmann, and Christian Scheideler. Anycasting in adversarial systems: routing and admission control. In *Proc. of the 30th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 1153–1168, 2003.
5. Yair Bartal. Distributed paging. In *Dagstuhl Workshop on On-line Algorithms*, pages 97–117, 1996.
6. Yair Bartal, Moses Charikar, and Piotr Indyk. On page migration and other relaxed task systems. *Theoretical Computer Science*, 268(1):43–66, 2001. Also appeared in *Proc. of the 8th SODA*, pages 43–52, 1997.
7. Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995. Also appeared in *Proc. of the 24th STOC*, pages 39–50, 1992.
8. Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *Proc. of the 44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 462–471, 2003.
9. Marcin Bienkowski. Dynamic page migration with stochastic requests. In *Proc. of the 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 270–278, 2005.
10. Marcin Bienkowski. *Page Migration in Dynamic Networks*. PhD thesis, Universität Paderborn, 2005.
11. Marcin Bienkowski and Jarosław Byrka. Bucket game with applications to set multicover and dynamic page migration. In *Proc. of the 13th European Symp. on Algorithms (ESA)*, pages 815–826, 2005.
12. Marcin Bienkowski, Mirosław Dynia, and Mirosław Korzeniowski. Improved algorithms for dynamic page migration. In *Proc. of the 22nd Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 365–376, 2005.
13. Marcin Bienkowski and Mirosław Korzeniowski. Dynamic page migration under brownian motion. In *Proc. of the European Conf. in Parallel Processing (Euro-Par)*, pages 962–971, 2005.
14. Marcin Bienkowski, Mirosław Korzeniowski, and Friedhelm Meyer auf der Heide. Fighting against two adversaries: Page migration in dynamic networks. In *Proc. of the 16th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 64–73, 2004.
15. Marcin Bienkowski and Friedhelm Meyer auf der Heide. Page migration in dynamic networks. In *Proc. of the 30th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 1–14, 2005. Invited paper.
16. David L. Black and Daniel D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.
17. L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78:243–263, 1997.
18. Marek Chrobak, Lawrence L. Larmore, Nick Reingold, and Jeffery Westbrook. Page migration algorithms using work functions. In *Proc. of the 4th Int. Symp. on Algorithms and Computation (ISAAC)*, pages 406–415, 1993.
19. R. Elsässer, M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. A Simple Graph-Theoretic Model for Selfish Restricted Scheduling. In *Proceedings of the 1st Workshop on Internet and Network Economics (WINE'05)*, to appear.
20. R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Nashification and the Coordination Ratio for a Selfish Routing Game. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP'03)*, Lecture Notes in Computer Science, Vol. 2719, Springer Verlag, pages 514–526, 2003.
21. R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Selfish Routing in Non-Cooperative Networks: A Survey. In B. Rován and P. Vojtás, editors, *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, Lecture Notes in Computer Science, Vol. 2747, Springer Verlag, pages 21–45, 2003.
22. C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: a computational study. *Mathematical Programming*, 81:229–256, 1998.
23. S. Fischer and B. Vöcking. On the Structure and Complexity of Worst-Case Equilibria. In *Proceedings of the 1st Workshop on Internet and Network Economics (WINE'05)*, to appear.
24. P.-O. Fjällström. Algorithms for graph partitioning: A survey. Linköping Electronic Articles in Computer and Information Science, 1998.

25. M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing Nash Equilibria for Scheduling on Restricted Parallel Links. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 613–622, 2004.
26. M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. The Price of Anarchy for Polynomial Social Cost. In J. Fiala, V. Koubek, and J. Kratochvíl, editors, *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS'04)*, Lecture Notes in Computer Science, Vol. 3153, Springer Verlag, pages 574–585, 2004.
27. M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and M. Rode. Nash Equilibria in Discrete Routing Games with Convex Latency Functions. In J. Diaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP'04)*, Lecture Notes in Computer Science, Vol. 3142, Springer Verlag, pages 645–657, 2004.
28. M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P. Spirakis. Extreme Nash Equilibria. In C. Blundo and C. Laneve, editors, *Proceedings of the 8th Italian Conference on Theoretical Computer Science (ICTCS'03)*, Lecture Notes in Computer Science, Vol. 2841, Springer Verlag, pages 1–20, 2003. Also accepted to *Theoretical Computer Science, Special Issue on Game Theory Meets Theoretical Computer Science*.
29. M. Gairing, T. Lücking, B. Monien, and K. Tiemann. Nash Equilibria, the Price of Anarchy and the Fully Mixed Nash Equilibrium Conjecture. In *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP'05)*, Lecture Notes in Computer Science, Vol. 3850, Springer Verlag, pages 51–65, 2005.
30. M. Gairing, B. Monien, and K. Tiemann. Selfish Routing with Incomplete Information. In *Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'05)*, pages 203–212, 2005.
31. E. Johnson, A. Mehrotra, and G. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993.
32. S. E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3):177–191, 2000.
33. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufman, 1992.
34. T. Lücking, M. Mavronicolas, B. Monien, and M. Rode. A New Model for Selfish Routing. In V. Diekert and M. Habib, editors, *Proceedings of the 21st International Symposium on Theoretical Aspects of Computer Science (STACS'04)*, Lecture Notes in Computer Science, Vol. 2996, Springer Verlag, pages 547–558, 2004.
35. T. Lücking, M. Mavronicolas, B. Monien, M. Rode, P. Spirakis, and I. Vrto. Which is the Worst-Case Nash Equilibrium? In B. Rován and P. Vojtás, editors, *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, Lecture Notes in Computer Science, Vol. 2747, Springer Verlag, pages 551–561, 2003.
36. Carsten Lund, Nick Reingold, Jeffery Westbrook, and Dicky C. K. Yan. Competitive on-line algorithms for distributed data management. *SIAM Journal on Computing*, 28(3):1086–1111, 1999. Also appeared as On-Line Distributed Data Management in *Proc. of the 2nd ESA*, pages 202–214, 1994.
37. R. D. McKelvey and A. McLennan. Computation of Equilibria in Finite Games. In H. Amman, D. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, 1996.
38. K. Mehlhorn and S. Nähler. LEDA: A Platform for Combinatorial and Geometric Computing. *Communications of the ACM*, 38(1):96–102, 1995.
39. J. F. Nash. Equilibrium Points in n -Person Games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.
40. J. F. Nash. Non-Cooperative Games. *Annals of Mathematics*, 54(2):286–295, 1951.
41. C. H. Papadimitriou. Algorithms, Games, and the Internet. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC'01)*, pages 749–753, 2001.
42. R. Preis and R. Diekmann. PARTY - A Software Library for Graph Partitioning. In B.H.V. Topping, editor, *Advances in Computational Mechanics with Parallel and Distributed Processing*, pages 63–71. Civil-Comp Press, 1997.
43. Rajmohan Rajaraman. Topology control and routing in ad hoc networks: a survey. *SIGACT News*, 33(2):60–73, 2002.
44. Theodore S. Rappaport. *Wireless Communications: Principles and Practices*. Prentice Hall, 1996.
45. T. Roughgarden. How Unfair is Optimal Routing. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 203–204, 2002.
46. Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. In *Proc. of the 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 489–500, 2004.
47. Christian Scheideler. Models and techniques for communication in dynamic networks. In *Proc. of the 19th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 27–49, 2002.
48. Christian Schindelhauer, Tamás Lukovszki, Stefan Rührup, and Klaus Volbert. Worst case mobility in ad hoc networks. In *Proc. of the 15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 230–239, 2003.
49. M. Sellmann, N. Sensen, and L. Timajev. Multicommodity flow approximation used for exact graph partitioning. In G. Di Battista and U. Zwick, editors, *Algorithms - ESA 2003*, volume LNCS 2832, pages 752–764, 2003.
50. N. Sensen. Lower Bounds and Exact Algorithms for the Graph Partitioning Problem Using Multicommodity Flows. In F. Meyer auf der Heide, editor, *Proceedings of the 9th Annual European Symposium on Algorithms (ESA'01)*, Lecture Notes in Computer Science, Vol. 2161, Springer Verlag, pages 391–403, 2001.
51. Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
52. A. Steger and N.C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probab. and Comput.*, 8:377–396, 1999.

53. J. G. Wardrop. Some Theoretical Aspects of Road Traffic Research. In *Proceedings of the Institute of Civil Engineers, Pt. II, Vol. 1*, pages 325–378, 1952.
54. Jeffery Westbrook. Randomized algorithms for the multiprocessor page migration. *SIAM Journal on Computing*, 23:951–965, 1994. Also appeared in *Proc. of the DIMACS Workshop on On-Line Algorithms*, pages 135–149, 1992.