Efficient schemes for nearest neighbor load balancing

Ralf Diekmann\textsuperscript{a,\!*1}, Andreas Frommer\textsuperscript{b,2}, Burkhard Monien\textsuperscript{a,1}

\textsuperscript{a} Department of Mathematics and Computer Science, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany
\textsuperscript{b} Department of Mathematics and Institute for Applied Computer Science, University of Wuppertal, D-42097 Wuppertal, Germany

Abstract

We design a general mathematical framework to analyze the properties of nearest neighbor balancing algorithms of the diffusion type. Within this framework we develop a new Optimal Polynomial Scheme (OPS) which we show to terminate within a finite number $m$ of steps, where $m$ only depends on the graph and not on the initial load distribution.

We show that all existing diffusion load balancing algorithms, including OPS, determine a flow of load on the edges of the graph which is uniquely defined, independent of the method and minimal in the $l_2$-norm. This result can also be extended to edge weighted graphs.

The $l_2$-minimality is achieved only if a diffusion algorithm is used as preprocessing and the real movement of load is performed in a second step. Thus, it is advisable to split the balancing process into the two steps of first determining a balancing flow and afterwards moving the load. We introduce the problem of scheduling a flow and present some first results on its complexity and the approximation quality of local greedy heuristics. © 1999 Published by Elsevier Science B.V. All rights reserved.

Keywords: Nearest neighbor balancing algorithms; Diffusion load balancing algorithms; Optimal Polynomial Scheme (OPS); Complexity; Local greedy heuristics

Partly supported by the DFG-Sonderforschungsbereich 376 Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen and the EC ESPRIT Long Term Research Project 20244 (ALCOM-IT).

\!* Corresponding author. Current address: Hilti AG, Corp. Research, FL 9494 Schaan, Principality of Liechtenstein.
1 http://www.upb.de/cs/ag-monien.html
2 http://www.math.uni-wuppertal.de/SciComp/

0167-8191/99/$ – see front matter © 1999 Published by Elsevier Science B.V. All rights reserved.

PII: S 0 1 6 7 - 8 1 9 1 ( 9 9 ) 0 0 0 1 8 - 6
1. Introduction

We consider the following abstract distributed load balancing problem. We are given an arbitrary, undirected, connected graph $G = (V,E)$ in which node $v_i \in V$ contains a number $w_i$ of unit-sized tokens. Our goal is to determine a schedule to move tokens across edges so that finally, the weight on each node is (approximately) equal. In each step we are allowed to move any number of tokens from a node to each of its neighbors in $G$. Communication between non-adjacent nodes is not allowed. We assume that the situation is fixed, i.e., no load is generated or consumed during the balancing process, and the graph $G$ does not change.

This problem describes load balancing in synchronous distributed processor networks and parallel machines when we associate a node with a processor, an edge with a communication link of unbounded capacity between two processors, and the tokens with identical, independent tasks [5]. It also models load balancing in parallel adaptive finite element simulations where a geometric space, discretized using a mesh, is partitioned into sub-regions and the computation proceeds on mesh elements in each sub-region independently [7,9]; here we associate a node with a mesh region, an edge with the geometric adjacency between two regions, and tokens with mesh elements in each region. As the computation proceeds, the mesh refines/coarsens depending on problem characteristics such as turbulence or shocks (in the case of fluid dynamics simulations, for example) and the size of the sub-regions (in terms of numbers of elements) has to be balanced. Because elements have to reside in their geometric adjacency, they can only be moved between adjacent mesh regions, i.e., via edges of the graph [7]. The problem of parallel finite element simulation has been extensively studied – see the book [9] for an excellent selection of applications, case studies and references.

Scalable algorithms for our load balancing problem operate locally on the nodes of the graph. They iteratively balance the load of a node with its neighbors until the whole network is globally balanced. The class of local iterative load balancing algorithms distinguishes between diffusion [3,5] and dimension exchange [5,21] iterations which mainly differ in the model of communication they are based on. Diffusion algorithms assume that a node of the graph is able to send and receive messages to/from all its neighbors simultaneously, whereas dimension exchange uses only pairwise communication, iteratively balancing with one neighbor after the other. Throughout this work we focus on diffusive schemes, i.e., we assume that nodes are able to communicate via all their edges simultaneously.

The quality of a balancing algorithm can be measured in terms of numbers of steps it requires to reach a balanced state and in terms of the amount of load moved over the edges of the graph. Recently, diffusive algorithms gained some new attention [6,7,11,15,17,20,21]. The original algorithm described by Cybenko [5] and, independently, by Boillat [3] lacks in performance because of its very slow convergence to the balanced state. Ghosh et al. use the idea of over-relaxation – a standard technique in numerical linear algebra – to speed up the iteration process by an order of magnitude [11]. We will see in the following that other, more advanced techniques from numerical linear algebra can be used to develop local
iterative methods showing an improved – and in a sense even optimal – convergence behavior.

Hu and Blake investigate the flow of load via the edges of the graph and propose a non-local method to determine a balancing flow which is minimal in the $l_2$-norm [15]. From experimental observations they conjecture that the local diffusion iteration of Cybenko also ends up with an $l_2$-minimal flow. We will see in the following that this is indeed the case, i.e., we give a mathematical proof that all local iterative diffusion algorithms including our new optimal OPS scheme determine a balancing flow which is $l_2$-minimal, uniquely defined and independent of the method and the parameters used.

Some of the more theoretical papers dealing with diffusion algorithms suggest to move the load directly as the iteration proceeds [5,11]. This one-phase approach usually moves load items back and forth over the edges as the iteration proceeds. Thus, the resulting flow of load is by far not $l_2$-optimal. In practice, therefore, the diffusion iteration is used as preprocessing just to determine the balancing flow. The real movement of load is performed in a second phase [7,15,17,20]. In addition, this two-phase approach has the advantage of avoiding any problems with adopting the local iterative algorithms to integral values (like it is, for example, done in [11]).

The movement of load has to be scheduled in such a way that each node does not send more load than it possesses in a certain step. Using experiments, we will see that simple greedy heuristics like they are used in practical applications like e.g. [20] allow to finish the load movement in much less steps than taken by the fastest diffusion algorithm. Interestingly, this flow scheduling problem appears to be un-studied up to now. So we introduce it here together with some first theoretical results.

The main contributions of this paper are summarized as follows:

- Based on matrix polynomials we develop a general mathematical framework to analyze the convergence behavior of existing diffusion type methods. Within this framework, we develop an Optimal Polynomial Scheme (OPS) which determines a balancing flow within $m$ steps if $m$ is the number of distinct eigenvalues of the graph. The OPS algorithm makes use of the full set of eigenvalues which can be computed in a preprocessing step. Information about the initial load distribution is not necessary.
- We consider the quality of the balancing flows determined by local iterative diffusion algorithms. We show that all such algorithms end up with the same flow of load which is optimal in the $l_2$-norm, provided the diffusion matrix is a scaled and shifted version of the Laplacian. We show how to extend this result to find minimal flows on edge-weighted graphs.
- We introduce the flow scheduling problem and discuss some general lower bounds for the number of steps needed to schedule $l_2$-minimal balancing flows. Additionally, we show that certain local greedy heuristics for this scheduling problem are $\Theta(\sqrt{n})$-optimal, and that all local greedy algorithms are $O(\sqrt{n})$-optimal.

The paper is organized as follows. Section 2 gives some basic definitions and notations. Section 3 develops the general framework for analyzing nearest neighbor schemes and presents the new optimal method. Section 4 shows that the methods
considered here all find \( l_2 \)-minimal flows of load. Section 5 deals with the flow scheduling problem and, finally, Section 6 shows results of some simulations.

## 2. Load balancing on graphs

### 2.1. Basic definitions and notations

Let \( G = (V, E) \) be a connected, undirected graph with \(|V| = n \) nodes and \(|E| = N \) edges. Let \( w_i \in \mathbb{R} \) be the load of node \( v_i \in V \) and \( w \in \mathbb{R}^n \) be the vector of load values. The vector \( \bar{w} := \frac{1}{n} (1, \ldots, 1) \) with \( \lambda = \sum_{i=1}^n w_i \) denotes the corresponding vector of average load.

Define \( A \in \{ -1, 0, 1 \}^{n \times N} \) to be the node-edge incidence matrix of \( G \). \( A \) contains a row for each node and a column for each edge. Each column has exactly two non-zero entries – a ‘1’ and a ‘-1’ – according to the two nodes incident to the corresponding edge. The signs of these non-zeros (implicitly) define directions for the edges of \( G \). These directions will later on be used to express the direction of the flow.

Let \( B \in \{ 0, 1 \}^{n \times n} \) be the adjacency matrix of \( G \). As \( G \) is undirected, \( B \) is symmetric. Column/row \( i \) of \( B \) contains 1s at the positions of all neighbors of \( v_i \). For some of our constructions we need the Laplacian \( L \in \mathbb{Z}^{n \times n} \) of \( G \) defined as \( L := A^T B - A \). This relation will be used extensively in Section 4.

Let \( x \in \mathbb{R}^N \) be a flow on the edges of \( G \). The direction of the flow is given by the directions in \( A \) in conjunction with the signs of the entries of \( x \), i.e., \( x_e > 0 \) denotes a flow in the direction of edge \( e \), \( x_e < 0 \) against. \( x \) is called a balancing flow on \( G \) if

\[
Ax = w - \bar{w}. \tag{1}
\]

Eq. (1) expresses the fact that the flow balance at each node corresponds to the difference between its initial load and the mean load value, i.e., after shipping exactly \( x_e \) tokens via each edge \( e \in E \), the load is globally balanced.

Among the set of possible flows which fulfill (1) we are interested in such \( x \) achieving certain quality criterions. We especially look at balancing flows \( x \) with minimal \( l_2 \)-norm defined as \( \|x\|_2 = \left( \sum_{i=1}^N x_i^2 \right)^{1/2} \).

By local iterative balancing algorithms we denote a class of methods performing iterations on the nodes of \( G \) which require communication with adjacent nodes only. The simplest of these methods performs on each node \( v_i \in V \) the iteration

\[
\forall e \in \{ v_i, v_j \} \in E : y_e^{k+1} = \alpha_e (w_i^{k+1} - w_j^{k+1}), \quad x_e^k = x_e^{k+1} + y_e^{k+1}, \quad \text{and} \quad w_i^k = w_i^{k+1} - \sum_{e=\{v_i,v_j\} \in E} y_e^{k+1}. \tag{2}
\]

Here, \( y_e^k \) is the amount of load sent via edge \( e \) in step \( k \). This scheme is known as the diffusion algorithm and has been described by Cybenko [5] and, independently, by Boillat [3]. Denoting by \( a = (\alpha_1, \ldots, \alpha_N)^T \) the vector of edge weights and by \( D = \text{diag}(a) \in \mathbb{R}^{N \times N} \) the \( N \times N \) diagonal matrix containing the edge weights on its
diagonal, (2) can be written in matrix notation as \( w^k = Mw^{k-1} \) with \( M = I - ADA^T \in \mathbb{R}^{n \times n} \). The non-zero structure of \( M \) is equivalent to the adjacency matrix \( B \). \( M \) contains \( x_e \) at position \((i,j)\) of edge \( e = (v_i, v_j) \), \( 1 - \sum_{e \in \{v_i,v_j\} \in E} x_e \) at diagonal entry \( i \), and 0 elsewhere. The edge weights \( x_e \) have to be chosen in such a way that \( M \) is nonnegative, i.e., all entries must be \( \geq 0 \). Hence, \( M \) is nonnegative, symmetric and doubly stochastic, i.e., its rows and columns sum up to 1. We call such a matrix \( M \) a diffusion matrix, if it has the additional property that in the case that the graph \( G \) is bipartite, at least one diagonal entry is positive. Then \(+1\) is a simple eigenvalue of \( M \) and all other eigenvalues are smaller in modulus \([5]\). Consequently, the iteration (2) converges to the average load \( \bar{w} \) \([5]\). If \( x_e = x \) for all edges \( e \in E \), i.e., all edge weights take the same value, then \( M \) is of the special form \( M = I - xL \). We will see in Section 4 that in this case the iteration Eq. (2) converges to an (uniquely defined) \( l_2 \)-minimal flow \( x \), independent of the value of \( x \). If \( M \) is of the more general form \( M = I - ADA^T \), iteration (2) determines a flow which is minimal in some weighted Euclidean norm. See Section 4.3 for details.

After a balancing flow has been computed, a schedule of load movements has to be found obeying the flow demands. This is particularly easy if initially each node has sufficiently many tokens to fulfill the demands on its outgoing edges. In this case the load can be balanced in one step. In the general case, a valid schedule has to be found obeying the flow demands. This is particularly easy if initially each node has sufficiently many tokens to fulfill the demands on its outgoing edges. In this case the load can be balanced in one step. In the general case, a valid schedule has to be found obeying the flow demands.

More formally, let \( \tilde{A} \in \{-1,0,1\}^{n \times N} \) be the incidence matrix \( A \) of \( G \) where the implicit edge directions express the directions of the flow, i.e., \( \tilde{x}_i = |x_i| \) for all \( i \) and \( \tilde{A}\tilde{x} = w - \bar{w} \). Let \( \tilde{A} = \tilde{A}^+ + \tilde{A}^- \) be a decomposition of \( \tilde{A} \) into its positive and negative part. With \( \tilde{A}^- \in \{-1,0\}^{n \times N} \) (\( \tilde{A}^+ \in \{0,1\}^{n \times N} \)) we denote the \( n \times N \) matrix derived from \( \tilde{A} \) by setting all the +1-entries (−1-entries) to 0. The flow scheduling problem is defined as follows:

**Definition 1 (The flow scheduling problem).** Input: A graph \( G \), node weights \( w^0 := w \), a flow \( \tilde{x} \) and a number \( k \geq 0 \). Question: Is there a decomposition \( S(\tilde{x}) = (\tilde{x}^0, \ldots, \tilde{x}^{k-1}) \) of the flow \( \tilde{x} \) with

\[
\tilde{x} = \sum_{j=0}^{k-1} \tilde{x}^j \quad \text{and} \quad w^{j+1} = w^j + \tilde{A}\tilde{x}^j = w^j + \tilde{A}^+ \tilde{x}^j \quad \text{(send)} + \tilde{A}^- \tilde{x}^j \quad \text{(receive)}
\]

such that

\[
w^j + \tilde{A}^- \tilde{x}^j \geq 0 \quad \forall j = 0, \ldots, k - 1.
\]

A schedule \( S(\tilde{x}) \) satisfying (4) is called valid. A valid schedule with minimal \( k \) among all possible valid schedules is called *time-optimal*.

Note that for this type of scheduling problem the weight \( \tilde{x}_e \) of an edge \( e \) determines how many tokens have to be send via this edge. However, the destination of a
token is not known in advance and has to be determined by the schedule. This problem is of interest in a much broader context than load balancing. It appears whenever a flow has to be scheduled and certain constraints have to be satisfied.

For the rest of the paper we separately deal with the two problems of finding a balancing flow and finding a schedule for the flow, i.e., we consider algorithms finding \( l_2 \)-optimal flows by local iterations in Sections 2.2 and 3, and the scheduling problem in Section 5. Note that because we propose to use the iterative algorithms just to determine the flow, it is possible for them to operate on real values.

2.2. Existing approaches

The problem considered here is a static version of dynamic load balancing where load items are generated and consumed continuously and balancing algorithms operate online. A variant of our problem where nodes are allowed to send only one token per step is known as the token distribution problem and has been studied extensively (see e.g. [10,16]). However, as we consider the more realistic model of multi-port communication on links of unbounded capacity, the token distribution results do not apply here.

A number of algorithms exist to solve the problem of nearest neighbor load balancing on graphs in the multi-port communication model. The earliest local method is the diffusive scheme (2) from [3,5]. It is sometimes also denoted as first order scheme (FOS) [11]. Its relatively slow convergence can be sped up by using the overrelaxed scheme \( w^k = \beta M w^{k-1} + (1 - \beta) w^{k-2} \) which is also called the second order scheme (SOS) [11]. Section 3.2 investigates the convergence properties of these methods in more detail. Their main advantage is their local nature, i.e. they exclusively use nearest neighbor communication on the edges of \( G \). One of the main contributions of [11] is the adaptation of the FOS and the SOS methods to the realistic setting of integral load values. For the SOS, the authors introduce so called IOUs to handle the case that processors have to send more load than they possess (which really happens during certain stages of SOS). With the splitting between balancing flow calculation and load movement we propose, these “integrality” problems do not appear.

Practical applications already use the diffusion methods for preprocessing only. The movement of load items is performed afterwards using greedy strategies [7,17,20]. Hu and Blake suggest to determine the balancing flow directly by solving a system of linear equations [15]. Their method explicitly finds \( l_2 \)-minimal flows, although the suggested use of a CG algorithm for the solution of the linear systems requires a lot of global communication and they need to know the average load \( \bar{w} \) in advance.

There exist some multi-level approaches to the balancing problem [14,18]. They recursively bisect the graph and balance in each step the load of the parts via the cut, thereby fixing the flow on the cut edges. Such algorithms terminate within \( \log n \) steps, where each step is quite complex and requires itself a lot of communication between processors.
3. Local and optimal local algorithms

In this section we present a general framework for nearest neighbor load balancing schemes which rely on a polynomial based representation of the iteratively determined work loads. We show that the FOS and SOS schemes appear as special cases within the framework as well as a scheme based on the Chebyshev polynomials. Moreover, we present OPS, a new polynomial scheme based on certain optimality conditions. This scheme determines the average load after a finite number of iterative steps. The numerical experiments reported in Section 6 show that OPS can significantly improve over the SOS method.

3.1. General framework

Let \( m \) be the number of distinct eigenvalues of \( M \). Since \( M \) is a diffusion matrix (see Section 2.1), \( 1 = \mu_1 > \mu_2 > \cdots > \mu_m > -1 \) are its eigenvalues, \( \mu_1 = 1 \) is a simple eigenvalue and \( (1, 1, \ldots, 1) \) is an eigenvector of \( M \) to eigenvalue \( \mu_1 \) [2]. We denote by \( \gamma = \max\{\vert \mu_2 \vert, \vert \mu_m \vert \} < 1 \) the second largest eigenvalue of \( M \) according to absolute values.

The following simple lemma is crucial to the analysis of any polynomial based scheme.

**Lemma 1.** Let \( w^0 \) be any initial work load and \( \overline{w} = \frac{1}{n}(1, \ldots, 1) \) with \( \lambda = \sum_{i=1}^{n} w_i^0 \) the corresponding average load. Moreover, let

\[
w^0 = \sum_{i=1}^{m} z_i
\]

be a representation of \( w^0 \) in terms of (not necessarily normalized) eigenvectors \( z_i \) of \( M \) where \( Mz_i = \mu_i z_i \), \( i = 1, \ldots, m \). Then

\[
\overline{w} = z_1.
\]

**Proof.** Of course, \( \overline{w} \) is an eigenvector of \( M \) with eigenvalue 1. Since 1 is a simple eigenvalue, we know that \( \overline{w} = az_1 \), \( a \neq 0 \). Denoting \( \langle \cdot, \cdot \rangle \) the Euclidean inner product on \( \mathbb{R}^n \) we get

\[
\langle w^0, \overline{w} \rangle = \sum_{i=1}^{m} \langle z_i, \overline{w} \rangle = \langle z_1, \overline{w} \rangle = \frac{1}{a} \langle \overline{w}, \overline{w} \rangle.
\]

Here, we made use of the fact that \( \overline{w} = az_1 \) is orthogonal to all other eigenvectors \( z_2, \ldots, z_m \). But now, \( \langle w^0, \overline{w} \rangle = \sum_{i=1}^{n} \frac{1}{n} w_i^0 = \frac{\lambda}{n} \) and \( \langle \overline{w}, \overline{w} \rangle = \sum_{i=1}^{n} \langle \frac{\lambda}{n}, \frac{\lambda}{n} \rangle = \lambda^2 / n \), so that Eq. (5) yields \( 1/a = 1 \). □

**Definition 2.** A polynomial based load balancing scheme is any scheme for which the work load \( w^k \) in step \( k \) can be expressed in the form

\[
w^k = p_k(M)w^0, \quad \text{where} \quad p_k \in \mathcal{P}_k.
\]
Here, $\mathcal{P}_k$ denotes the set of all polynomials $p$ of degree $\deg(p) \leq k$ satisfying the constraint $p(1) = 1$.

Note that the condition $p_k(1) = 1$ implies that all row sums in the matrix $p_k(M)$ are equal to 1. This in turn means that the total work load is conserved, i.e., $\sum_{i=1}^{n} w_i^k = \sum_{i=1}^{n} w_i^0$. Let us also note that the representation (6) is useful primarily for mathematical analysis. Indeed, for (6) defining an algorithmically feasible nearest neighbor scheme, it must be possible to rewrite it as an update process where $w^k$ is computed from $w^{k-1}$ (and maybe some more previous iterates) involving one multiplication with $M$ only. This means that the polynomials $p_k$ have to satisfy some kind of short recurrence relation. Such relations will explicitly be stated in the special cases to be discussed in the following subsections.

The convergence of a polynomial based scheme depends on whether (and how fast) the ‘error’ $e_k = w^k - \overline{w}$ between the iterate $w^k = p_k(M)w^0$ and the corresponding average load $\overline{w} = \frac{\lambda}{n}(1, \ldots, 1)$ with $\lambda = \sum_{i=1}^{n} w_i^0$ tends to zero. These errors $e_k$ have two fundamental properties which we state in Lemma 2.

**Lemma 2.** Let $w^0 = \sum_{i=1}^{m} z_i$ as in Lemma 1. Then

$$e^0 = \sum_{i=2}^{m} z_i,$$

$$e^k = p_k(M)e^0, \quad k = 0, 1, 2, \ldots$$

**Proof.** Since $w^0 = e^0 + \overline{w}$, the first equality is a direct consequence of Lemma 1. To show (8) we note that due to $p_k(1) = 1$ the vector $\overline{w}$ is an eigenvector of $p_k(M)$ with eigenvalue 1. This yields

$$e^k = w^k - \overline{w} = p_k(M)(w^0 - \overline{w}) = p_k(M)e^0. \quad \square$$

### 3.2. FOS, SOS, and Chebyshev

Using both statements from Lemma 2 we see that the error $e^k$ of any polynomial based scheme satisfies

$$e^k = p_k(M) \left( \sum_{i=2}^{m} z_i \right) = \sum_{i=2}^{m} p_k(M)z_i = \sum_{i=2}^{m} p_k(\mu_i)z_i.$$

Here we made use of $p_k(M)z_i = p_k(\mu_i)z_i$, since $z_i$ is an eigenvector of $M$ with eigenvalue $\mu_i$. This fundamental relation allows to analyze several nearest neighbor load balancing schemes in detail. In particular, taking the Euclidean norm and observing that the $z_i$ are orthogonal, we arrive at

$$\|e^k\|_2^2 = \sum_{i=2}^{m} p_k(\mu_i)^2 \|z_i\|_2^2 \leq \left[ \max_{i=2}^{m} p_k(\mu_i)^2 \right] \sum_{i=2}^{m} \|z_i\|_2^2,$$
which, since \( \|e^0\|_2^2 = \sum_{i=2}^{m} \|z_i\|_2^2 \), yields
\[
\|e^k\|_2 \leq \max_{i=2}^{m} |p_k(\mu_i)| \|e^0\|_2.
\] (10)

We start our analysis of different methods with the FOS scheme of Cybenko [5] (cf. Section 2.1), where we have \( p_k(t) = t^k \). These polynomials satisfy the simple short recurrence \( p_k(t) = tp_{k-1}(t) \), \( k = 1, 2, \ldots \), so that we get
\[
w^k = Mw^{k-1}, \quad k = 1, 2, \ldots
\]
In this situation \( |p_k(\mu_i)| = |\mu_i^k| \leq \gamma^k \) for \( i = 2, \ldots, m \), where \( \gamma = \max_{i=2}^{m} |\mu_i| \). Thus, (10) gives
\[
\|e^k\|_2 \leq \gamma^k \|e^0\|_2.
\]

The second order scheme SOS of [11] takes the polynomials
\[
p_0 \equiv 1, \quad p_1(t) = t, \quad p_k(t) = \beta tp_{k-1}(t) + (1 - \beta)p_{k-2}(t), \quad k = 2, 3, \ldots
\]
so that
\[
w^1 = Mw^0, \quad w^k = \beta Mw^{k-1} + (1 - \beta)w^{k-2}, \quad k = 2, 3, \ldots
\] (11)

Here, \( \beta \) is a fixed parameter. This scheme is known as the second order Richardson method in numerical analysis. Investigations in [13] show that this iteration converges to \( \overline{w} \) whenever \( \beta \in (0, 2) \) and that the fastest convergence occurs for
\[
\beta = \beta_{opt} = 2 \left/ \left( 1 + \sqrt{1 - \gamma^2} \right) \right.
\] (12)

In this case, one has (see [13,19])
\[
\max_{t \in [-\gamma, \gamma]} |p_k(t)| = (\beta_{opt} - 1)^{k/2} \left( 1 + k\sqrt{1 - \gamma^2} \right).
\]

Since \( \max_{i=2}^{m} |p_k(\mu_i)| \leq \max_{t \in [-\gamma, \gamma]} |p_k(t)| \) we therefore get from (10)
\[
\|e^k\|_2 \leq (\beta_{opt} - 1)^{k/2} \left( 1 + k\sqrt{1 - \gamma^2} \right) \|e^0\|_2.
\] (13)

As was pointed out in [11], when comparing the factors \( \gamma^k \) of FOS and \( (\beta_{opt} - 1)^{k/2} (1 + k\sqrt{1 - \gamma^2}) \) of SOS for \( \gamma \) close to 1, one can interpret this as the SOS method being of ‘second’ order whereas FOS is only ‘first’ order.

The Chebyshev method differs from SOS only by the fact that the parameter \( \beta \) will now depend on \( k \) according to
\[
\beta_1 = 1, \quad \beta_2 = \frac{2}{2 - \gamma^2}, \quad \beta_k = \frac{4}{4 - \gamma^2\beta_{k-1}}, \quad k = 3, 4, \ldots
\] (14)

The corresponding polynomials \( p_k \) are the (scaled) Chebyshev polynomials for the interval \([-\gamma, \gamma]\). This means that they are optimal in the sense that (see [13,19])
Similarly to SOS, this yields the estimate
\[ \| e^k \|_2 \leq \frac{2 \left( \beta_{\text{opt}} - 1 \right)^{k/2}}{1 + \left( \beta_{\text{opt}} - 1 \right)^2} \| e^0 \|_2. \] (15)

The factor in (15) is always smaller than in (13) which shows that the Chebyshev method is usually to be preferred over the SOS scheme. Asymptotically, however, both methods can be regarded to perform identically since
\[ \lim_{k \to \infty} \frac{2 \left( \beta_{\text{opt}} - 1 \right)^{k/2}}{1 + \left( \beta_{\text{opt}} - 1 \right)^2} \] 
\[ \overset{!}{=} \lim_{k \to \infty} \left( \beta_{\text{opt}} - 1 \right)^{k/2} \left( 1 + k \sqrt{1 - \gamma^2} \right) \] 
\[ = \left( \beta_{\text{opt}} - 1 \right)^{1/2}. \]

### 3.3. Optimal polynomial methods

The basic estimate (10) suggests to construct a method where the quantity
\[ \max_{i=2} \max_{t \in [-\gamma, \gamma]} |p_i(t)| \]
is minimized over all polynomials from \( P_k \). Unfortunately, this would not result in short recurrences between the polynomials, so that we will not get a computationally viable nearest neighbor scheme. However, as we will now explain, minimizing the quantity
\[ \sum_{i=2}^m p_i(\mu_i)^2 \]
will give us adequate recurrences. Accordingly, the idea of the method to be developed now is to obtain the smallest possible factor in the estimate
\[ \| e^k \|_2 \leq \left( \sum_{i=2}^m p_i(\mu_i)^2 \right) \max_{i=2} \| z_i \|_2^2, \] (16)
which follows from (9) in a trivial manner. We need some additional terminology. For any two polynomials \( p, q \) we define the (indefinite) inner product \( \langle \cdot, \cdot \rangle \) as
\[ \langle p, q \rangle := \sum_{j=2}^m \omega_j p(\mu_j) q(\mu_j), \]
where \( \omega_2, \ldots, \omega_m \) are a priori given positive weights. Note that \( \langle p, p \rangle \) is always non-negative and \( \langle p, p \rangle = 0 \) if and only if \( p(\mu_j) = 0 \), \( j = 2, \ldots, m \). In particular, \( \langle p, p \rangle > 0 \) for all polynomials \( p \in \Pi_{m-2} \), since the \( m \) constraints \( p(1) = 1, p(\mu_i) = 0, i = 2, \ldots, m \) cannot be simultaneously satisfied for polynomials of degree \( \leq m - 2 \).
We are interested in polynomials $p_k$ which minimize $\langle p, p \rangle$ over $\Pi_k$. The following theorem gives a rather complete answer.

**Theorem 1.** For $k = 0, \ldots, m - 1$ define the polynomials $p_k \in \Pi_k$ as follows:

\[
p_0(t) \equiv 1, \quad p_1(t) = \frac{1}{\gamma_1} [(x_1 - t)p_0(t)],
\]

\[
p_k(t) = \frac{1}{\gamma_k} [(x_k - t)p_{k-1}(t) - \beta_k p_{k-2}(t)], \quad k = 2, \ldots, m - 1,
\]

where

\[
\begin{align*}
\alpha_k = \langle p_{k-1}, p_{k-1} \rangle / \langle p_{k-1}, p_{k-2} \rangle, & \quad k = 1, \ldots, m - 1, \\
\beta_k = \gamma_{k-1} \langle p_{k-1}, p_{k-1} \rangle / \langle p_{k-2}, p_{k-2} \rangle, & \quad k = 2, \ldots, m - 1, \\
\gamma_1 = \alpha_1 - 1, & \quad \gamma_k = \alpha_k - \beta_k, \quad k = 2, \ldots, m - 1.
\end{align*}
\]

Then we have

\[
\langle p_k, p_j \rangle = 0 \quad \text{for } k, j = 0, \ldots, m - 1, \quad k \neq j
\]

and

\[
\sum_{j=2}^{m} \left( \frac{\omega_j}{1 - \mu_j} \right) p_k(\mu_j)^2 = \min_{p \in \Pi_k} \sum_{j=2}^{m} \left( \frac{\omega_j}{1 - \mu_j} \right) p(\mu_j)^2, \quad k = 0, \ldots, m - 1.
\]

**Proof.** Basically, the whole theorem is known from numerical analysis since it states the main properties of the (scaled and shifted) so-called kernel polynomials with respect to $\langle \cdot, \cdot \rangle$; see [8, Sec. 2.5], e.g. For convenience, we reproduce the main parts of a proof, here.

The relation (19) means that the polynomials $p_k$ are orthogonal with respect to $\langle \cdot, \cdot \rangle$. As is well known from numerical analysis [8], such a sequence of orthogonal polynomials exists and it is unique up to scalings with a scalar factor. This scalar factors are uniquely defined for our situation, since we have the additional restriction $p_k(1) = 1$ for $k = 0, \ldots, m - 1$. (Note that it is also known that the orthogonal polynomials have all their zeros within the interval $[\mu_\alpha, \mu_\beta]$, so that none of them vanishes at $t = 1$.) Finally, the recurrence (17) is just the standard three term recurrence for orthogonal polynomials (see again [8]), adapted to the normalization $p_k(1) = 1$.

To show (20) let us first introduce the notation $\langle \cdot, \cdot \rangle_0$ for the inner product

\[
\langle p, q \rangle_0 = \sum_{j=2}^{m} \left( \frac{\omega_j}{1 - \mu_j} \right) p(\mu_j) q(\mu_j),
\]

so that $\langle p, q \rangle = \langle p, (1 - t)q \rangle_0$. Now, let $p \in \Pi_k$. Since $p - p_k$ has a zero at $t = 1$, we see that $p(t)$ can be represented as $p(t) = p_k(t) - (1 - t)q(t)$ with $\deg(q) \leq k - 1$. Moreover, any polynomial $q$ of degree $\leq k - 1$ can be represented as a linear combination $q(t) = \sum_{j=0}^{k-1} \eta_j p_j(t)$. Therefore, we have
\[ \langle p, p \rangle_0 = \langle p_k + (1 - t)q, p_k + (1 - t)q \rangle_0 \]
\[ = \langle p_k, p_k \rangle_0 + 2\langle p_k, (1 - t)q \rangle_0 + \langle (1 - t)q, (1 - t)q \rangle_0 \]
\[ = \langle p_k, p_k \rangle_0 + (1 - t)q, (1 - t)q \rangle_0, \]
(21)
since the term \( \langle p_k, (1 - t)q \rangle_0 = \langle p_k, q \rangle = \sum_{j=0}^{k-1} \eta_j \langle p_k, p_j \rangle \) vanishes due to (19). If \( q \) with \( \deg(q) \leq k - 1 \) is not identically zero, the quantity \( \langle (1 - t)q, (1 - t)q \rangle_0 \) is positive. So (21) shows that \( p_k \) is indeed the unique minimizer of \( \langle p, p \rangle \) for \( p \in \mathcal{P}_k \). \( \square \)

Taking \( \omega_j = 1 - \mu_j \), Theorem 1 shows how to construct a sequence of polynomials \( p_k \) for which the bound \( \sum_{j=2}^{m} p_k(\mu_j)^2 \) from (16) is the smallest possible. Turning this into a computational algorithm, we realize that we first have to precompute all eigenvalues of the matrix \( M \) and then precompute the scalars \( \alpha_i, \beta_i, \gamma_i \) from Theorem 1. Once this is done, and the scalars \( \alpha_i, \beta_i, \gamma_i \) are made available to all processors, we get the optimal polynomial nearest neighbor load balancing scheme OPS:

\[ w^1 = \frac{1}{\tilde{\gamma}_1} [x_1 w^0 - M w^0] \]
\[ w^k = \frac{1}{\gamma_k} [x_k w^{k-1} - M w^{k-1} - \beta_k w^{k-2}], \quad k = 2, \ldots, m - 1. \]

Note that \( p(t) = \prod_{j=2}^{m} (1 - t/\mu_j) \) is the only polynomial from \( \mathcal{P}_{m-1} \) which achieves \( \langle p, p \rangle = 0 \), i.e. \( p_{m-1}(t) = p(t) \). Thus, (16) gives \( \omega^{m-1} = 0 \), which shows that the above method is a finite method in the sense that it arrives at \( w^k = \overline{w} \) in at most \( k = m - 1 \) steps. Let us note that the standard CG method [12] shares this finite termination property. However, the CG method requires the computation of two inner products within each iterative step, so it is not a local method.\(^3\)

4. Solution quality

The purpose of this section is to show that the load balancing algorithms of Section 3.3 can easily be modified in such a manner that, in addition to the iterative work loads \( w^k \), they also compute an \( l_2 \)-minimal flow from \( w^0 \) to \( w^k \). These modifications represent only minor additional cost. In particular, no further communication (neither global nor local) is required. The essential assumption is that the diffusion matrix \( M \) in the load balancing scheme is of the form

\[ M = I - zL, \]
(22)
where \( L \) is the Laplacian of the processor graph and \( z \) is a fixed weight for all edges \( e \in E \). In this case, the flow \( x \) transforming \( w^0 \) into \( w^k \) which is determined by the local iterative methods is uniquely defined and \( l_2 \)-minimal. The more general form of \( M = I - ADA^T \) will be discussed in Section 4.3.

\(^3\) As was pointed out by an anonymous referee, the inner products in CG can however be avoided if all eigenvalues and the first components of the eigenvectors of \( M \) are known.
We first collect some basic results on $l_2$-minimal flows. We then proceed by showing how to modify the FOS, SOS, Chebyshev, and OPS schemes of Section 3.3 so that they compute $l_2$-minimal flows together with the iterative work loads.

4.1. Basic results

For a graph $G = (V, E)$ let $A \in \{-1, 0, 1\}^{n \times N}$ be its incidence matrix and $L = AA^T \in \mathbb{Z}^{n \times n}$ its Laplacian as defined in Section 2.1. Our goal is to characterize $l_2$-minimal flow solutions for given work loads $w$ and $v \in \mathbb{R}^n$, i.e., vectors $x \in \mathbb{R}^n$ which have minimal norm $\|x\|_2$ under all those satisfying $Ax = b$ where $b = w - v$.

We start with a lemma recalling an elementary fact about the image of the linear map defined by $L$. Throughout the whole section, $\langle \cdot, \cdot \rangle$ will always denote the Euclidean inner product on $\mathbb{R}^n$.

**Lemma 3.** The equation $Lz = b$ has a solution (and then infinitely many), if and only if $b \in \tilde{w}^\perp$. Here, the orthogonal complement $\tilde{w}^\perp$ denotes the space of all vectors $y$ perpendicular to $\tilde{w}$, i.e., $\langle y, \tilde{w} \rangle = 0$.

**Proof.** It is well-known (cf. e.g. [4]) that the Laplacian of a connected graph has 0 as a simple eigenvalue, the corresponding eigenspace being spanned by $\tilde{w}$. As for any symmetric matrix, the image of $L$ is precisely the orthogonal complement of its kernel. □

We are now able to state the following characterization of $l_2$-minimal flows.

**Lemma 4.** Consider the $l_2$ minimization problem

$$\text{minimize} \|x\|_2 \quad \text{over all } x \text{ with } Ax = b.$$  

Provided that $b \in \tilde{w}^\perp$, the solution to this problem is given by

$$x = A^Tz, \quad \text{where } Lz = b. \quad (23)$$

**Proof.** This lemma has been proved in [15] using Lagrange multipliers. Here is a very elementary proof: First, note that by Lemma 3 we actually know that the second equation in (23) does have a solution. Of course, $x$ from (23) satisfies $Ax = b$. Any other $y$ which satisfies $Ay = b$ can thus be written as $y = x + v$, where $Av = 0$. We have

$$\|y\|_2^2 = \|x\|_2^2 + 2\langle x, v \rangle + \|v\|_2^2,$$

where $\langle x, v \rangle = \langle A^Tz, v \rangle = \langle z, Av \rangle = 0$, so that $\|y\|_2$ is indeed minimal if and only if $v = 0$. □

Lemma 5 shows that if we have a sequence of work loads converging to the average load, and if we have an $l_2$-minimal flow for each such load, then these minimal flows converge to the minimal flow for the average load.
Lemma 5. Let $w^k$ be a (finite or infinite) sequence of work loads which converges to the average load $\bar{w}$. Moreover, let
\[ w^k = w^0 + Ax^k \]
be such that $\|x^k\|_2$ is minimal, i.e., (by Lemma 4)
\[ x^k = A^Tz^k, \quad \text{where } Lz^k = w^k - w^0. \]
Then, $\lim_{k \to \infty} x^k = \bar{x}$ exists, $\bar{w} = w^0 + A\bar{x}$, and $\|\bar{x}\|_2$ is minimal.

Proof. Let $\lambda = \sum_{i=1}^n w_i^n$ so that $\bar{w}_i = \lambda/n$, $i = 1, \ldots, n$. Note first that by Lemma 3 the equation $Lz^k = w^k - w^0$ does have a solution since $\langle w^k, \bar{w} \rangle = (\lambda/n) \sum_{i=1}^n w_i^n = \bar{w}^2/n = \langle w^0, \bar{w} \rangle$, i.e., $w^k - w^0 \in \bar{w}$. Now, since there are several $z^k_i$ satisfying $Lz^k = w^k - w^0$, let us take the (Moore–Penrose [12]) pseudoinverse solution for all $k$, i.e.,
\[ z^k = L^\dagger(w^k - w^0). \]
This immediately implies that $\lim_{k \to \infty} z^k = \bar{z}$ exists, satisfying $\bar{z} = L^\dagger(\bar{w} - w^0)$. Consequently, $\lim_{k \to \infty} x^k = \bar{x}$ exists, too, and it satisfies $\bar{x} = A^T\bar{z}$ as well as $\bar{w} = w^0 + A\bar{x}$.
So, by Lemma 4, $\bar{x}$ is the $l_2$-minimal flow. 

Hu and Blake suggest to solve $Lz = \bar{w} - w^0$ directly using for example the conjugate gradient iteration [15]. The flow is then given as $x = A^Tz$. We show in the following how to iteratively update $x$ within any of the nearest neighbor schemes considered so far such that $x^k$ converges to the $l_2$-minimal flow $\bar{x}$. In this manner we get a true nearest neighbor scheme for computing the minimal flow as well.

4.2. Computing work loads and minimal flows

We start with a general observation which holds for any polynomial based method with diffusion matrix $M$, i.e., for methods where we have $w^k = p_k(M)w^0$ with $p_k \in \Pi_k$. Since $p_k(1) = 1$, the polynomial $p_k(1 - xt)$ has value 1 for $t = 0$, so that we get the representation
\[ p_k(1 - xt) = 1 + tq_{k-1}(t), \quad \deg(q_{k-1}) \leq k - 1. \]
Because of (22) this shows that $w^k = p_k(M)w^0 = w^0 + Lq_{k-1}(L)w^0$, so that $z^k$ from Lemma 5 is given by $q_{k-1}(L)w^0$. Thus, the $z^k$ and, consequently, the $x^k$ are related in quite a straightforward manner to the polynomials defining the load balancing method. However, for practical algorithmic formulations we have to turn this relation into a cheap update process for the $x^k$. Thus, our goal is to find a vector $d^{k-1}$ which is easy to update and which can be used to calculate a flow increment $\delta^{k-1}$. The next theorem describes the update process. Note that similar results are quite familiar in numerical analysis in the context of iterative methods for linear systems, see [8], e.g.

Theorem 2. Let $p \in \Pi_k$ be a polynomial satisfying the 3-term recurrence relation
\[ p_k(t) = (\sigma_k t - \tau_k)p_{k-1}(t) + p_k p_{k-2}(t) \] (24)
with
\[ \sigma_k - \tau_k + \rho_k = 1 \quad \text{for all } k = 1, 2, \ldots \] \tag{25}

Let
\[ d^0 = -x^0, \quad x^1 = y^0 = A^T d^0, \quad w^1 = w^0 + Ay^0, \]
and for \( k = 2, 3, \ldots \)
\begin{align*}
d^{k-1} &= -x\sigma_k w^{k-1} - \rho_k d^{k-2}, \\
x^k &= x^{k-1} + y^{k-1}, \\
w^k &= w^{k-1} + A y^{k-1},
\end{align*} \tag{26}
be the update process for \( x^k \) and \( w^k \). Then, \( \lim_{k \to \infty} x^k = \bar{x} \) and \( \lim_{k \to \infty} w^k = \bar{w} \) exist, \( \bar{w} = w^0 + \bar{A} \bar{x} \) and \( ||x||_2 \) is minimal.

**Proof.** With \( p_k(1 - \alpha t) = 1 + tq_{k-1}(t) \) Eq. (24) becomes
\[ 1 + tq_{k-1}(t) = (\sigma_k(1 - \alpha t) - \tau_k)(1 + tq_{k-2}(t)) + \rho_k(1 + tq_{k-3}(t)) \]
which, after some algebraic manipulations, yields
\[ q_{k-1}(t) = -x\sigma_k(1 + tq_{k-2}(t)) + (\sigma_k - \tau_k)q_{k-2}(t) + \rho_k q_{k-3}(t) \]
This shows that \( z^k \) from Lemma 5 is given by
\[ z^k = -x\sigma_k w^{k-1} + (\sigma_k - \tau_k)z^{k-1} + \rho_k z^{k-2}. \]
Substituting \( z^k = z^{k-1} + d^{k-1} \) and \( z^{k-2} = z^{k-1} - d^{k-2} \) and using (25) we finally arrive at the update formula
\[ d^{k-1} = -x\sigma_k w^{k-1} - \rho_k d^{k-2}. \tag{27} \]
for \( d^{k-1} \). Theorem 2 now follows with Lemma 5. □

Looking at the schemes discussed so far, we have for the FOS \( \sigma_k = 1, \tau_k = \rho_k = 0 \) so that \( d^{k-1} = -xw^{k-1} \). In the Chebyshev scheme, for each but the first step, we have \( \sigma_k = \beta_k, \tau_k = 0, \rho_k = (1 - \beta_k) \) which yields \( d^{k-1} = -x\beta_k w^{k-1} - (1 - \beta_k)d^{k-2} \). The first step is identical to FOS. The SOS scheme differs from Chebyshev only by the fact that \( \sigma_k = \beta_{\text{opt}} \). Finally, for all but the first step in the OPS scheme we have \( \sigma_k = 1/\gamma_k, \tau_k = -x_k/\gamma_k \) and \( \rho_k = -\beta_k/\gamma_k \) so that (27) results in \( d^{k-1} = (xw^{k-1} + \beta_k d^{k-2})\gamma_k \). The first step can be formulated in a similar way. Fig. 1 shows a general frame for the different types of diffusive load balancing. This local update scheme of the form
\[ w^k_i = w^{k-1}_i - \sum_{e=\{(n,e)\in E}} y^{k-1}_e \quad \text{with} \quad y^{k-1}_e = \begin{cases} \sigma_k x_e (w^{k-1}_e - w^{k-1}_i) & \text{if } k = 1 \\ \sigma_k x_e (w^{k-1}_e - w^{k-1}_i) - \rho_k y^{k-2}_e & \text{if } k \geq 2 \end{cases} \]
is equivalent to (24) which can be shown by induction using (25) (cf. the proof to Lemma 3 in [11]).
Note that all schemes discussed here do not need to know \( \bar{w} \) in advance as opposed to, for example, Hu and Blake’s method [15]. All the more advanced schemes like SOS, Chebyshev or OPS need is (at least some partial) information on the eigenvalues of the graph \( G \).

### 4.3. Weighted \( l_2 \)-norms

In situations where the costs of moving load via edges of \( G \) are not homogeneous, we are interested in a minimal flow with respect to a weighted Euclidean norm, i.e. solutions \( \hat{x}^k \) of the problem

\[
\text{minimize } \| \hat{x}^k \|_c = \left( \sum_{i=1}^{N} c_i (\hat{x}^k)^2 \right)^{1/2} \quad \text{over all } \hat{x}^k \text{ with } A\hat{x}^k = w^k - \bar{w}. \tag{28}
\]

Here, \( c = (c_1, \ldots, c_N) \) is a cost vector for the edges with all positive entries. An example for such a situation arises with graphs representing a quotient graph of a finite element triangulation where for geometric reasons flows on certain edges are preferred to others. This can be modeled by attributing different cost factors \( c_i \) to the edges [7].

We would now like to sketch that this weighted least squares problem can be treated in a manner completely analogous to what we have presented in the previous paragraphs. To this purpose, let \( C \) denote the \( N \times N \) diagonal matrix with \( C_{ii} = \sqrt{c_i}, \ i = 1, \ldots, N \). Define \( \hat{x}^k = Cx^k \) and \( \check{A} = AC^{-1} \). Then the weighted least squares problem (28) is equivalent to the unweighted problem

\[
\text{minimize } \| \hat{x}^k \|_2 \quad \text{over all } \hat{x}^k \text{ with } \check{A}\hat{x}^k = w^k - \bar{w}. \tag{29}
\]

Let us define the weighted Laplacian \( \check{L} \) as \( \check{L} = \check{A}\check{A}^T = AC^{-2}A^T \). A careful inspection of Lemmas 3–5 now shows that they remain valid with these new matrices. In particular, the solution of (29) can be computed as
\[ \bar{x}^k = \bar{A}^T \bar{z}^k \text{ where } \bar{L} \bar{z}^k = w^k - w^0. \]

In a manner completely similar to our previous investigations, we can thus show that polynomial schemes for load balancing can be modified such that they compute \( \bar{z}^k \) along with the work loads \( w^k \), provided that the diffusion matrix \( \bar{M} \) is now of the form

\[ \bar{M} = I - z\bar{L} = I - zAC^{-2}A^T. \]

For purposes of practical computation, the only modifications to be done to the explicit algorithms is to replace \( M \) by \( \bar{M} \), to rename \( x_k \) as \( \bar{x}_k \) and to include the back transformation \( x_k = C^{-1}\bar{x}_k \).

As another interpretation of this result we see that a diffusive scheme using the more general form of \( M = I - AD_{ii}A^T \) with a non-negative diagonal matrix \( D = \text{diag}(a_1, \ldots, a_N) \) (cf. Section 2.1) yields a weighted \( l_2 \)-minimal flow with associated cost vector \( c = (1/a_1, \ldots, 1/a_N) \).

### 5. Flow scheduling

Section 4.3 developed methods to determine a balancing flow for a given graph \( G = (V, E) \) and initial load situation \( w^0 \). We now consider the question of actually moving the load and here especially the problem of scheduling the flow such that no node sends more than it possesses and the number of steps is minimized.

The flow scheduling problem has been defined formally in Section 2.1. We will first give some examples showing that the two-step approach of first determining the flow and afterwards moving the load is superior to a one-step approach moving the load directly. Then, in Section 5.2 we show a general lower bound on the time a schedule of a \( l_2 \)-minimal balancing flow can take. Finally, Section 5.3 discusses the quality of local greedy scheduling heuristics. In the following, we assume that the matrix \( A \) is directed according to the flow, i.e., we omit the tilde of \( \bar{A} \) and \( \bar{x} \) of Section 2.1.

#### 5.1. Two-step versus direct load movement

If diffusive algorithms are used for direct load movement, they typically shift much more tokens than necessary. We demonstrate this by considering the simple example of a chain of three nodes \( u, v, w \) as shown in Fig. 2. The left node \( u \) and the right node \( w \) hold \( 3r \) tokens, is initially empty. For the flow, assume that the edges are (implicitly) directed towards \( v \). Consider the FOS algorithm with parameter \( z = 1/2 \). The accumulated flow on both edges is given by

\[ x_e = -3r \sum_{i=1}^{\infty} \left( -\frac{1}{2} \right)^i = 3r \left( 1 - \frac{2}{3} \right) = r. \]
whereas, summing up the amounts gives
\[ \hat{x}_r = 3r \sum_{i=1}^{\infty} \frac{1}{2^i} = 3r(2 - 1) = 3r. \]

So we see that even in this small example the one-step approach would move a factor three more load than necessary.

Let us now consider the time needed for scheduling the flow in the two-phase approach. Experimental observations show that usually the load can be moved in a small number of steps after the balancing flow has been found. Fig. 5 (Section 6) shows impressive examples for this fact: all nearest neighbor schemes take a rather large number of iterative steps (even the optimal one), whereas the load movement using a simple greedy strategy (as defined in Section 5.3) is finished after at most three steps.

We conjecture that any nearest neighbor load balancing algorithm has to require at least as many steps as an optimal schedule based on a flow determined by the same local algorithm. However, it remains an open problem how to prove this observation.

Finally, let us observe that rounding up the flows at edges to integral values does not introduce arbitrarily large errors. Let \( \tilde{x} \) be the balancing flow rounded to integral values and \( \tilde{w} \) the load distribution after moving \( \tilde{x} \). The largest difference to the setting in \( \mathbb{R} \) occurs if at a node for each incoming edge \( e \) we have \( \tilde{x}_e = x_e + 1/2 \) and for each outgoing edge \( \tilde{x}_e = x_e - 1/2 \) (or vice versa). Thus, for all nodes \( v_i \in V \) it holds \( |\tilde{w}_i - \tilde{w}_j| \leq \frac{1}{2} \deg(v_i) \).

### 5.2. Lower bounds on \( k \)

It is interesting to notice that the diameter of the graph is not an upper bound on the number of steps a schedule can take. More specifically, we can show that there exist graphs with \( n^2 + 1 \) nodes and diameter \( O(1) \) where any scheduling of an \( l_2 \)-optimal balancing flow has to take at least \( \frac{1}{4}n - 1 \) steps.

Consider the graph \( G \) shown in Fig. 3. It consists of \( n \) levels, each containing \( n \) nodes. The levels are connected by complete bipartite graphs, the bottom-node of each level is connected to the special node \( v \) which is a kind of ‘short-cut’. The
diameter of this graph is 4. Assume that the leftmost level holds all the load, \((n + 1)r\) for the bottommost node \(u\) and \(nr\) tokens for the other nodes from the first level. Then, \(\bar{w} = r\) is the average load.

Let \(x\) be a flow sending \(r\) tokens from \(u\) to \(v\) and moving the rest from left to right via the bipartite graphs using the edges between two consecutive levels evenly. Then,

\[
\frac{\|x\|^2}{r^2} = 1 + \sum_{i=1}^{n-1} n^2 \left( \frac{n-i}{n} \right)^2 = \frac{1}{6} n(n-1)(2n-1) < \frac{1}{3} n^3 \quad \text{for } n \geq 3.
\]

Now assume there is a flow \(\tilde{x}\) which is schedulable in \(k\) steps. If \(k < n\), then \((n - 1 - k)\) levels of \(G\) have to receive their load via node \(v\). Thus, the amount of \((n - 1 - k)nr\) tokens have to be transferred towards \(v\) via at most \(k - 2\) edges and have to leave \(v\) via at most \((n - 1 - k)\) edges. Distributing the load over the available edges to and from \(v\) yields

\[
\frac{\|\tilde{x}\|^2}{r^2} \geq (k - 2) \left( \frac{(n - 1 - k)n}{k - 2} \right)^2 + (n - 1 - k) \left( \frac{(n - 1 - k)n}{n - 1 - k} \right)^2 = \frac{n^2(n-1-k)(n-3)}{k-2},
\]

where we have counted only the flow on edges to and from \(v\). (30) is not larger than \(\frac{1}{3} n^3\) only if \(k \geq \frac{3}{4} n - 1\). Thus, any \(l_2\)-minimal flow must take at least \(k \geq \frac{3}{4} n - 1\) steps.

5.3. Local greedy scheduling

A local greedy flow scheduling algorithm determines for each node and each step how many of the available tokens to send to which of the outgoing edges. Local greedy heuristics can be characterized by the following points:

(i) Their scheduling decision depends only on local information about the flow demands \(x\) and the available load \(w\).

(ii) If in a certain step a node contains enough tokens to fulfill all its outflow-demands, it immediately saturates all its outgoing edges.
(iii) If a node does not contain enough load, it distributes all available tokens to its outgoing edges according to some tie-breaking.

In the experiments reported in Section 6 such a local scheduling heuristic balances the load in only a small number of steps. We show that this is not always the case.

Let us first consider the class of memory-less greedy algorithms where a decision depends only on the current situation and not on the history. By Round–Robin Greedy (RRG), we denote the local greedy scheduling algorithm which fills up one edge after the other. Per step, RRG still moves as much load as possible, i.e., it sends all its available tokens, but chooses a subset of edges which are filled up to saturation (where the last edge in the subset might not be saturated completely). In contrast, Proportional Parallel Greedy (PPG) denotes the local greedy schedule which shifts load via all edges of a node in parallel and the amount is chosen proportional to the current demand of the edges. The following lemma shows that the RRG scheduling algorithm is $\Theta(\sqrt{n})$-optimal.

Lemma 6. For every balancing flow the Round–Robin local greedy scheduling algorithm is $\Theta(\sqrt{n})$-optimal.

Proof. Let $G = (V,E)$ be a graph with $|V| = n$ nodes, balancing flow $x$ and edge directions according to $x$. Assume that the RRG algorithm determines a schedule $S_R(x)$ of length $|S_R(x)| = g$ and that an optimal schedule $S_o(x)$ for $x$ requires $|S_o(x)| = o$ steps. We show that $g/o = \Theta(\sqrt{n})$.

$g \geq c_1 \sqrt{n}$: Consider the construction of Fig. 4. Let $p$ be the length of the upper line of nodes (the “backbone”). The graph contains $n = \frac{1}{2} p (p-1)$ nodes. Using the optimal schedule $S_o(x)$, the nodes on the backbone send their load to their right neighbor first. Thus, $o = |S_o(x)| = 3$. For the RRG algorithm we may assume that it sends the available load downwards first. In this case, the leftmost node of the backbone has to fulfill all the $p$ flow demands of the backbone which requires $p$ steps.

$g \leq c_2 \sqrt{n}$: Let $r$ be the average load per node and $\|x\|_1 = \sum_{i=1}^{N} x_i$ be the total amount of flow. Together with the greedy property (iii) of local heuristics we can make the following observations:

1. Tokens remain in a node only if the node has fulfilled all its outflow demands.
2. Tokens which end up in a node after $\delta$ rounds of scheduling have been moved over $\delta$ edges and, because $G$ is a DAG, have visited $\delta + 1$ different nodes (including first and last node on their path).

Fig. 4. The RRG local greedy heuristic is $\Omega(\sqrt{n})$-optimal.
3. Assume a flow $x$ on $G$ is realized within $\delta$ steps. Because each token can pass at most $\delta$ edges, at least $\lceil |x|_1 / \delta \rceil$ disjoint tokens have to be moved.

Consider now a schedule $S_g(x)$ determined by RRG. As $S_g(x)$ requires $g$ steps, there has to be at least one token traveling a path of length $g$. Let $(v_0, v_1, \ldots, v_{g-1}, v_g)$ be this path. As $v_{g-1}$ sends the token to node $v_g$ not before step $g$, it can receive its own $r$ tokens not before step $g-1$. Each of this tokens has traveled a distance of $g-1$. In the same way we see that each node $v_i$ on the path receives its own $r$ tokens not before step $i$. As all nodes on $(v_0, v_1, \ldots, v_{g-1}, v_g)$ are disjoint, so are all tokens remaining in the nodes on this path. Summing up the paths of these tokens we get

$$\|x\|_1 \geq r \sum_{i=1}^{g-1} i = \frac{r}{2} (g^2 - g).$$

As the optimal schedule $S_o(x)$ finishes the token movement within $o$ steps, the observation 3 from above requires the existence of at least $|x|_1/o$ tokens. Each node receives $r$ tokens, so there have to be

$$n \geq \frac{|x|_1}{ro} \geq \frac{g^2 - g}{2o}$$

nodes. If $o \geq 2$, (31) results in

$$g \leq 1 + o \sqrt{n}. \quad \square$$

Note that the upper bound did not make any assumptions about the scheduling strategy of the greedy heuristic and, thus, applies to all local greedy algorithms including RRG and PPG. The lower bound reduces to $\Omega(\log n)$ if PPG is used. Thus, for this algorithms a gap between the upper and lower bound shows up.

6. Experimental results

Here, we shortly report some numerical results which demonstrate the advantages of the optimal polynomial approach and compare it with the FOS and SOS schemes. The Chebyshev scheme behaves almost identical to SOS, so we do not reproduce data for that scheme. We also include experimental results for the greedy schedules RRG/PPG which, on these test instances, do not differ from each other.

We consider four different processor graphs, each with a total of 64 processors: The one-dimensional torus, the two-dimensional torus, the 6-dimensional hypercube, and a quotient graph arising from a partition of the finite element mesh “airfoil1” into 64 sub-domains [6]. For all examples, the initial work load is identically generated as a uniformly random distribution. All computations are performed with real numbers according to the algorithms given in Section 5.3. After the flow is determined, we round it up to integral values and schedule it using the local greedy heuristic from Section 5.3 The tie-breaking rule fills up outgoing edges proportional to their remaining demand.
For the flow calculation, the diffusion matrix $M$ is initially taken to be of the form $M = I - \alpha L$ with $L$ the Laplacian and $\alpha$ the inverse of the maximum node degree ($\alpha = 1/2, 1/4, 1/6$ and $1/10$ for the 1D Torus, the 2D Torus, the Hypercube and the FE-quotient graph, respectively). For the FOS and SOS scheme we apply an additional spectral shift of the form $M \leftarrow (1 - \delta)I + \delta M$ as described in [5] in order to minimize $\gamma$ and therefore maximize the speed of convergence (note that this shift also ensures $\gamma < 1$).

Fig. 5 shows that on average the OPS scheme requires only half as many iterations as SOS, with FOS being by far the slowest scheme (see also [11]). It is also apparent that in early iterations the SOS and the OPS schemes can behave quite similarly. Fig. 5 very clearly illustrates the fact that the OPS achieves the solution after $m - 1$ steps ($m$: number of different eigenvalues of $M$). Very interestingly, this convergence takes place quite 'brutally' with the immediately preceding iterates still being relatively far from the solution. Note that $m = 33, 13$ and $7$ for the 1D Torus, the 2D Torus and the Hypercube, respectively. Also, note that for any one-dimensional torus and for any hypercube $m - 1$ just equals the diameter of the graph.

Fig. 5 also shows that the time needed to actually balance the load if a valid balancing flow is known is much less than the number of iterations performed by any of the nearest neighbor schemes. On the four examples shown here, the simple greedy schedule requires $3, 2, 1$, and $2$ steps, and this appears to be a typical behavior for many other examples we tested.

Fig. 5. Performance of different balancing schemes ($\|w^k - \tilde{w}\|$ vs. iteration step $k$).
7. Conclusions

We have presented a general framework for analyzing diffusive nearest neighbor load balancing algorithms on graphs and developed an optimal polynomial balancing scheme (OPS). After a certain amount of preprocessing, OPS is guaranteed to determine a balancing flow within $m$ steps if $m$ is the number of distinct eigenvalues of the graph. For arbitrary diffusive nearest neighbor schemes we have shown how they can be modified to determine $l_2$-optimal balancing flows and we also extended the $l_2$-optimality criterions to edge weighted graphs.

We have shown that it is advisable to split the task of load balancing into two phases, first determine a balancing flow and second move the load. This maintains the $l_2$-optimality of the flow which is destroyed if load is moved directly. For the final movement of the load, we have introduced the flow scheduling problem. We showed that simple local greedy heuristics for the problem of scheduling a balancing flow are $\Theta(\sqrt{n})$-optimal and that arbitrary local heuristics are $O(\sqrt{n})$-optimal. Open up to now is the question of how to close the gap between the upper bound of $\sqrt{n}$ and the lower bound of $\log n$ for certain local scheduling strategies.

Acknowledgements

The authors thank Rainer Feldmann, Marco Riedel, Walter Unger, Rolf Wanka (Paderborn) and Peter Brucker (Osnabrück) for helpful discussions and suggestions. The proof of the upper bound in Theorem 6 was found by Marco Riedel (Paderborn). Helpful comments of an anonymous referee are also gratefully acknowledged.

References


