Dynamic Load Balancing by Random Matchings

Bhaskar Ghosh*
Department of Computer Science, Yale University, P.O. Box 208285, New Haven, Connecticut 06520
and
S. Muthukrishnan†
DIMACS, P.O. Box 1179, Rutgers University, Piscataway, New Jersey 08855

Received December 22, 1995

1. INTRODUCTION

Consider the following scenario in a distributed setting. An application program is running on a parallel or distributed network consisting of a large number of processors connected together in an arbitrary topology. Each processor has a load of independent tasks to be executed. This distribution of tasks is dynamically determined, that is, the specific application program running on the machine cannot be developed with a-priori estimates of the load distribution. The goal of dynamic load balancing is to reallocate the tasks so that each processor has nearly the same amount of load. Of course, in natural settings the scenario is more demanding in that the tasks might be dynamically generated or consumed in each step and additionally, the underlying topology might change owing to failures in communication links. Besides load balancing, scenarios such as the one above occur in several other guises, for example, in job scheduling, adaptive mesh partitioning and resource allocation problems. In each of these guises load redistribution is critical for the efficiency of algorithms.

Existing models for dynamic load balancing make one or more assumptions regarding communication in the underlying network. The focus of this paper concerns the degree of locality and the amount of parallelism in the models. Specifically,

1. Some existing models [AA+93, LM93, R91, C89] overestimate available parallelism in transferring load to neighbors by assuming that load can be moved from each processor to all its neighbors in parallel in each time step. However, for a large number of machines there are hardware limitations because of which the communication between a processor and its neighbors is inherently sequential. For example, in the Intel Paragon [PR94] all messages from the same processor have to pass sequentially through one network interface chip before being sent to neighbors via multiple ports. Also, in the IBM SP-1 and SP-2 machines [S+94, SP2], the messages have to traverse a single physical link sequentially to a network switching unit.

2. Several existing models [AA+93, LM93] underestimate available parallelism in link capacity by assuming that only one load unit can be transferred across a link at a time. This assumption overlooks a significant point. Communication links in most parallel computers (e.g. Intel iPSC/860 and Paragon [R94], IBM SP-1 [S + 94]) have large latency (order of 0.1 msec to 1 msec) and large bandwidth (order of 10 to 200MBytes/sec). For this reason, a common approach to reduce communication costs is to send few long messages rather than several short ones. There are large classes of important load balancing problems (e.g. fine grain functional programming [GH89], game tree searching [F93] and adaptive mesh partitioning [W91]) where the load units or tasks that have to be moved...
are of small size (order of tens of bytes). In such applications, several tasks or load units can be packaged together to be sent as one long message and thus make better use of the communication bandwidth.

3. A few existing models allow non-local load movement [LM93] and global control [E+86, LK87]. However, global communication and routing are expensive on most parallel and distributed computers. Furthermore, algorithms which rely on global information become even more expensive while adjusting to link failures.

Motivated by these observations, we study dynamic load balancing on distributed networks under a new model of load movement which we call the Matching Model. In the matching model, load can be moved only across a matching set of links in each step, that is, each processor is involved in transferring load with at most one neighbor. We also assume that each link has unbounded capacity, that is, any number of load units can be moved across a link in each step. Finally, we require our algorithms to use only local information and employ only local load movement. To solve the dynamic load balancing problem on our model, we study an abstract problem which we call progressive load scheduling (referred to as PLS henceforth). This problem is also of independent interest in job scheduling. Our main result is an asymptotically optimal algorithm for the PLS problem which we use to derive an efficient algorithm for the dynamic load balancing problem. Our algorithm works on networks of arbitrary topology which possibly undergo link failures during its execution. Its running time is related to the eigenstructure of the underlying graph.

The rest of the paper is organized as follows. We review the preliminaries and formally state our results in Section 2. In Section 3 we present our main result, namely the asymptotically optimal algorithm for the PLS problem. We use this to derive an algorithm for dynamic load balancing in Section 4. In Section 5 we present a summary of our experimental results analyzing issues in load balancing related to our algorithms.

2. PRELIMINARIES

2.1. Technical Preliminaries

Consider a graph $G$ with $n$ nodes with maximum degree $d$. Given a weight distribution $\mathbf{w} = (w_1, \ldots, w_n)^T$ on the nodes of $G$ where node $i$ has weight $w_i$, the potential $\phi$ of the graph is

$$\phi = \left(\sum_{i=1}^n w_i^T\right)^2 - n\mathbf{w}^2 = \sum_{i=1}^n (w_i - \mathbf{w})^2$$

where $\mathbf{w} = \sum w_i/n$ is the average load on a node. $\phi$ as defined is the square of the Euclidean distance between $\mathbf{w}$ and the vector $\mathbf{w}_{avg} = (\mathbf{w}, \mathbf{w}, \ldots, \mathbf{w})^T$ in which the total weight is equally distributed among all the nodes. Clearly $\phi \geq 0$ for any $\mathbf{w}$. Note that $\phi = 0$ if and only if $\mathbf{w} = \mathbf{w}_{avg}$.

We use the following simple linear algebraic concepts in this paper. Background on this can be found in [MP92]. Let $A$ denote the adjacency matrix of $G$. Let $D$ be the matrix $(d_{ij})$ where $d_{ij}$ is the degree of node $i$ if $i = j$, and is 0 otherwise. The matrix $L = D - A$ is the Laplacian Matrix of $G$. The eigenvalues of $L$ are $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$. The eigenvalue $\lambda_2$ is a widely studied parameter and it reflects the connectivity of the graph. (See [A86, MP92] for more on this.) We use the following well known facts about the eigenvalue $\lambda_2$, found, for example, in [MP92].

**Fact 1.** $G$ is a connected graph if and only if $\lambda_2 > 0$.

**Fact 2.** From the Courant–Fischer Minimax Theorem it follows that

$$\lambda_2 = \min_{x \not= 0} \left( \frac{\mathbf{x}^T L \mathbf{x}}{\mathbf{x}^T \mathbf{x}} \right)$$

where $\mathbf{u} = (1, 1, \ldots, 1)^T$, and $\mathbf{x} \not= 0$ is the eigenvector corresponding to $\lambda_2$, and $\mathbf{x}^T \mathbf{u}$ denotes $\mathbf{x}$ is orthogonal to $\mathbf{u}$.

In the rest of the paper, we use $E(X)$ to denote the expected value of a random variable $X$.

2.2. Our Model

Our machine model is a connected network of identical processors interconnected by communication links forming an arbitrary topology. Each processor has a load of unit-sized tasks to be executed. Tasks are identical and independent of each other; therefore, they can be scheduled to run on any processor. We assume that the processors work in lock-step, that is, there is a global clock. We also assume that between time steps processors can do any amount of computation. In our algorithms, the amount of computation is kept low.

Processors communicate by sending messages along the links. The links are bidirectional, which implies that processors connected by a link can send a message to each other simultaneously. Each time step can be a communication step or a load movement step. In a communication step, each processor can either send a message to one of its neighbors (at each processor, the incoming messages are queued) or read any one of the messages in its queue. In the load movement step, load is moved along only a set of links that form a matching (referred to as PLS henceforth). This problem is also of independent interest in job scheduling. Our main result is an asymptotically optimal algorithm for the PLS problem which we use to derive an efficient algorithm for the dynamic load balancing problem. Our algorithm works on networks of arbitrary topology which possibly undergo link failures during its execution. Its running time is related to the eigenstructure of the underlying graph.

...
movement once started on a live link is guaranteed to finish successfully by the end of that time step. When a link fails, we assume that the processor at its endpoints know about it immediately.

We call this the Matching Model.

2.3. Problems and Their Significance

We introduce and study the following abstract problem.

The Progressive Load Scheduling (PLS) Problem. Given a connected graph \( G = (V, E) \) with maximum degree \( d \) and an assignment \( w_i \) of integral weights to the nodes, the problem is to determine a set \( M \) of matching edges and for each edge in \( M \), a relocation of integral portions of the weights on its ends across that edge, so that the potential \( \phi \) of the graph is reduced.

We are interested in developing an algorithm for this problem on our machine model. For this we use the natural correspondence between \( G \) and our model with nodes representing processors and edges representing links. The load on processor \( i \) is the weight assigned to node \( i \). In the rest of the paper we use the words nodes/processors, edges/links and weight/load interchangeably.

Convergence Ratio. Given \( G \) and \( w \) and any algorithm for the PLS problem, let the potential before and after the invocation of the algorithm be \( \phi \) and \( \phi' \) respectively. The decrease in potential \( (\phi - \phi') \) is denoted by \( \Delta \phi \) and the convergence ratio of this algorithm is defined to be \( \frac{\Delta \phi}{\phi} \).

The Dynamic Load Balancing Problem. Given an assignment \( w \) of integral loads to the processors in our machine model, the problem is to redistribute the load so that a load-balanced state is reached. A load-balanced state is one in which \( |w_i - w_j| \leq 1 \) if \( w_i \) and \( w_j \) are the loads at processors \( i \) and \( j \) respectively that are connected by a link.

Note that for the same initial load assignment, there are several final load-balanced states with this property. It suffices for us to reach one such state. Recall that the loads are necessarily integral.

Significance of the PLS Problem. It is easy to see that dynamic load balancing problem can be solved by repeatedly invoking any algorithm for the PLS problem until no further drop in the potential is possible. This is because if the load is not balanced, PLS can be applied to further decrease the potential \( \phi \). To see this consider processors \( i \) and \( j \) connected by a link with \( |l_i - l_j| \geq 2 \). Without loss of generality, let \( l_j > l_i \). Then, moving one load unit from \( i \) to \( j \) across link \((i,j)\) decreases \( \phi \) by

\[
I_j^2 + I_j^2 - (I_i - 1)^2 - (I_j + 1)^2 = 2(I_i - I_j - 1) > 2.
\]

Thus when no further decrease in \( \phi \) is possible, \( |l_i - l_j| \leq 1 \) for all links \((i,j)\); that is, a load-balanced state has been reached.

Other than being useful for studying dynamic load balancing, the PLS problem turns out to be of independent interest in job scheduling. Consider a distributed program execution in which jobs are generated and consumed at various processors arbitrarily. In order to increase the throughput of the machine, a common practice is to interleave this execution with steps that schedule available jobs to underloaded or idle processors. Broadly, there are two known paradigms for this scheduling. One paradigm [KR89] guarantees that each processor has at least one job to execute at the end of scheduling. The other paradigm [C89] guarantees that all processors have roughly the same number of jobs at the end of the scheduling step. It is easy to see that in both these paradigms, there exist sequences of load generation and consumption that force any algorithm either to resort to load movement directly between two non-neighboring processors in one step (e.g., in the first paradigm) or long sequences of load movements between neighboring processors (e.g., in the second paradigm). Thus, scheduling turns out to be expensive. For more on these two existing paradigms, see [LK87, E + 86, NX + 85, Sta84].

Since these two known paradigms for job scheduling are expensive, we introduce an alternate paradigm of restricting algorithms to perform load movement only between neighbors, but requiring a guarantee of reasonable progress toward the load-balanced state. In our case, the reasonable progress is a decrease in the distance to the load-balanced state, where the distance is formalized by the potential function \( \phi \). This is precisely the PLS problem. As we show later, we present an efficient algorithm for the PLS problem which provides an algorithm for job scheduling under our paradigm.

2.4. Past Work

Dynamic load balancing has been studied in a number of settings. Almost all research has focused on algorithms for specific topologies and/or rely on global routing phases. A class of such research has involved performance analysis of load balancing algorithms by simulations [LMR91]. Among analytical results, load balancing on specific topologies under statistical assumptions on input load distributions has been studied [HCT89]. For arbitrary initial load distributions, load balancing has been studied in specific topologies such as Counting Networks [AHS91, HLS92], Hypercubes [P89], Meshes [HT93] and Expanders [PU89]. These algorithms do not extend to arbitrary or dynamically changing topologies. For dynamically changing topologies, load balancing has been studied...
Our Results and Our Approach

We present a local randomized algorithm for the PLS problem such that $E(\delta \phi|\phi) \geq \lambda_2/16d$ when $\phi$ is sufficiently large. This algorithm is asymptotically optimal since we show that no algorithm (even one which has global information and which is randomized) can guarantee a larger convergence ratio for all input graphs and weight distributions. For our algorithm, the final imbalance $A_f$ can be at most $D$ (in contrast, $A_f \leq dD$ in [AA + 93]). This is because our algorithm stops making progress when the difference in the load on the endpoints of any edge is at most $d$; thus, the maximum difference between any two processors in the network is at most $dD$.

Cygienko [C89] considers a model for load movement similar to ours but additionally allows each processor to transfer load to all its neighbors in one time step. We call his model as the multiport model. He makes the convenient (but unrealistic) assumption that the loads are real and therefore divisible to arbitrary precision. He presents a local load balancing algorithm and gives necessary and sufficient conditions for which his algorithm converges on arbitrary graphs. Later (in Section 4) we will compare our bounds with the ones in [C89].

2.5. Our Results and Our Approach

We proceed by first giving an algorithm for the case when there are no link failures in the underlying graph (Section 3.1) and then extend it to the case of possible link failures in the graph (Section 3.2).

3. ALGORITHMS FOR THE PLS PROBLEM

In this section we present our main technical result which is an asymptotically optimal algorithm for the PLS problem. We first give an algorithm for the case when there are no link failures in the underlying graph (Section 3.1) and then extend it to the case of possible link failures in the graph (Section 3.2).

3.1. PLS without Edge Failure

We proceed by first giving an algorithm for the case of real weights and then extend it to the case when the weights are integral.

3.1.1. Algorithm LR with Real Weights. Recall the PLS problem from Section 2.3 and for intuition consider solving the problem with real weights and without edge failures. Given a graph where the weight on some node is not equal to $w$, we can always pick an edge $(i, j)$ where $w_i \neq w_j$, and equalize the weights on its endpoints. Note that a single edge is trivially a matching and that equalizing the weights makes use of the assumption that edges have unlimited capacity. Equalizing the weights provably decreases $\phi$ since the reduction in potential $\delta \phi = (w_i^2 + w_j^2) - 2(w_i + w_j)/2y$, which is $2(w_i - w_j)^2 > 0$. Intuitively, we would expect that choosing a matching consisting of several edges and equalizing along each edge in the matching would result in a larger reduction $\delta \phi$. This however depends on how the set of matching edges is chosen; some choice of several edges in a matching might result in smaller $\delta \phi$ than that due to a single well-chosen edge. A set of matching edges can be obtained in several ways. For example, edge-coloring the input graph gives us a set of matchings where each color defines a matching. Alternately, given a graph we can explicitly compute the matching which gives the maximum potential drop. All these schemes require expensive computation of global information; also, they may not work when some edges disappear.
In our algorithm (Algorithm LR), we choose a random set of matching edges locally. The manner in which the random matching is chosen ensures that there is a global lower bound on the probability of each edge appearing in the matching. This property ensures global convergence bounds. For choosing such a random matching, we draw upon the intuition from the very sparse phase in the evolution of random graphs [87] as explained later.

The rest of this section is organized as follows. Algorithm LR for real weights is described in Figures 1 and 2. Its convergence properties are analyzed in Lemma 2 and Theorem 1 and the optimality of its convergence ratio is proved in Lemma 3, 4 and 5.

Description of Algorithm LR. Algorithm LR in Fig. 1 has two high level steps. In the Matching Step a random matching \( M \) is chosen locally. At the end of this step, if edge \((i, j)\) belongs to \( M \), then both endpoints \( i \) and \( j \) know that \((i, j)\) is in \( M \). In the Balancing Step, load movement is performed across the edges in \( M \).

It is easy to see how the Balancing Step can be implemented on our machine model. In the first step, each \( i \) such that \((i, j) \in M \) sends a message to \( j \) containing the value of \( w_i \). In the second step, each \( i \) such that \((i, j) \in M \) and \( w_i > w_j \) sends \((w_i - w_j) / 2\) load units in a message to \( j \). That completes the load balancing. Clearly the Balancing Step takes one communication step and one load movement step.

It remains for us to describe the implementation of the Matching Step in our machine model. The details are given in Fig. 2.

**Lemma 1.** In the implementation (Fig. 2) of Algorithm LR on our model, any link \((i, j)\) picked in \( S \) in Step 1 is in \( M \) if and only if there does not exist a link \((i, k)\) or \((j, k)\) in \( S \), where \( k \neq i \) and \( k \neq j \).

**Proof.** Assuming claim X that follows Step 3, it is clear that those links \((i, j)\) chosen to be in \( M \) in Steps 4 and 5 have the following property: \( j \) is a partner of \( i \), \( i \) is a partner of \( j \), and \( i \) and \( j \) are marked \( A \). Therefore in order to prove the lemma it suffices to prove claim X and the following that we denote claim Y: for any link \((i, j)\) in \( S \) after Step 1 where \( i \) is a partner of \( j \) and \( j \) is a partner of \( i \), processors \( i \) and \( j \) are both marked \( A \) if and only if there is no link \((i, k)\) in \( S \) and no link \((j, k)\) in \( S \) where \( k \neq i \) and \( k \neq j \).

Claim X is easy to see. Suppose processor \( i \) is marked \( A \) at the end of Step 3. Then there is some link \((i, j)\) in \( S \) picked by either \( i \) or \( j \) or both in Step 1. In any case, \( i \) gets a partner either in Step 2 (when it sends a message to \( j \)) or in Step 3 (when the only message in its queue is from \( j \)). So, \( i \) has a partner \( j \).

Now we prove claim Y. For one direction, assume that for any link \((i, j)\) in \( S \) after Step 1, \( i \) is a partner of \( j \), \( j \) is a partner of \( i \), processors \( i \) and \( j \) are both marked \( A \). So, \( i \) was not marked \( W \) either in Step 1 or 3. So there cannot be some link \((i, k)\) picked in \( S \) by either \( i \) or \( k \) where \( k \neq j \). A similar proof holds for \( j \). That proves one direction of the claim. For the other direction assume that there is \((i, j)\) picked in \( S \), \( j \) is a partner of \( i \), \( i \) is a partner of \( j \), and that there is no link \((i, k)\) in \( S \) or \((j, k)\) in \( S \) where \( k \neq j \). Since there is no link \((i, k)\) in \( S \) (where \( k \neq j \)) picked by either \( i \) or \( k \), \( i \) cannot be marked \( W \) in either Step 1 or 3. Hence it stays marked \( A \) at the end of Step 3. A similar proof holds for \( j \). Therefore, the other direction of the claim \( Y \) follows as well.

That completes the proof of the two claims and hence the lemma.

**Remark.** Distributed/Parallel algorithms for determining the maximal [186, 86] matchings of a given graph work by iteratively adding a random matching to a current matching. The manner in which a random matching is chosen there in each such step seems somewhat similar to our Matching Step.

**Analysis of Algorithm LR.** Recall that Step \( \alpha \) is the Matching Step in Algorithm LR (Fig. 1).
LEMMA 2. For each edge \((i, j)\) in \(G\),
\[
\Pr[(i, j) \in M \text{ at the end of Step } \mathcal{A}] \geq 1/8d
\]
where \(d\) is the maximum degree of graph \(G\).

Proof. Since both \(i\) and \(j\) could choose edge \((i, j)\) in Step \(\mathcal{A}\),
\[
\Pr[(i, j) \in S \text{ after Step } \mathcal{A}] = \frac{1}{8d_d} + \frac{1}{8d_j} - \frac{1}{64d^2}
\]
Let \(P_i\) denote the probability that at least one incident edge \((i, k)\) (where \(k \neq j\)) is chosen in \(S\) in Step \(\mathcal{A}\) by either \(i\) or \(k\). \(P_i\) is defined similarly. We upper bound \(P_i\) as follows:
\[
P_i \leq \sum_{k \neq j} \Pr[(i, k) \in S \text{ after Step } \mathcal{A}] \\
= \sum_{k \neq j} \frac{1}{4d_{ik}} \leq \frac{d_i - 1}{4 \min d_{ik}} \leq \frac{d_i - 1}{4d_i}
\]
Therefore,
\[
\Pr[(i, j) \in S \text{ after Step } \mathcal{A} \text{ and removed in Step } \mathcal{A}] \\
\leq \Pr[(i, j) \in S \text{ after Step } \mathcal{A}] (P_i + P_j) \\
\leq \left(\frac{1}{4d_d} - \frac{1}{64d^2}\right)(d_i - 1) + \frac{1}{4d_j} + \frac{1}{4d_i}
\leq \frac{1}{8d_d} + \frac{1}{16d_d d_d} - \frac{1}{16d_d d_d}
\leq \frac{1}{8d_d} + \frac{1}{8d^2}
\]
Finally,
\[
\Pr[(i, j) \in M \text{ after Step } \mathcal{A}] \\
= \Pr[(i, j) \in S \text{ after Step } \mathcal{A}] \text{ and it is not removed in Step } \mathcal{A} \\
= \Pr[(i, j) \in S \text{ after Step } \mathcal{A}] \\
- \Pr[(i, j) \in S \text{ after Step } \mathcal{A} \text{ and removed in Step } \mathcal{A}] \\
\geq \left(\frac{1}{4d_d} - \frac{1}{64d^2}\right) - \left(\frac{1}{8d_d} - \frac{1}{8d^2}\right) \text{ (by inequality 1)}
\geq \frac{1}{8d}\]

Since the probability that each edge is picked in the matching \(M\) is at least \(1/8d\), we expect to see roughly \((nd/2)/8d = n/16\) edges in \(M\) out of a total of at most \(nd/2\) edges. If we consider a random graph with edge probability \(1/4n\) which corresponds to graphs with roughly \(n/8\) edges, then most connected components in the graph are isolated edges or small trees [B87]. We have shown in the proof of Lemma 2 that removing the small trees still leaves us with enough isolated edges forming a matching of size roughly \(n/16\).

THEOREM 1. On applying Algorithm LR on any connected graph \(G\) with load distribution \(\bar{w}\) and potential \(\psi\), we have \(E(\delta \psi)/\delta \psi \geq \tilde{\alpha}/2\).

Proof. Let \(M\) be the matching chosen by Algorithm LR. For each edge \((i, j)\) in \(E\), let \(\delta \phi_{i,j}\) denote the decrease in potential by equalizing the weights on nodes \(i\) and \(j\) if \((i, j) \in M\). Clearly \(\delta \phi_{i,j} = w^2_{i} + w^2_{j} - 2(w_i + w_j)/2 = (w_i - w_j)^2/2\).

\[
E(\delta \phi) = \sum_{(i,j) \in E} \Pr[(i,j) \in M] \times (\delta \phi_{i,j}) \\
= \sum_{(i,j) \in E} \Pr[(i,j) \in M] \times \frac{(w_i - w_j)^2}{2} \\
\geq \frac{1}{16d} \sum_{(i,j) \in E} (w_i - w_j)^2 \\
= \frac{1}{16d} \sum_{(i,j) \in E} (w_i - w_j)^2
\]
Note that \(\phi = (\sum_i w_i^2) - m\bar{w}^2 = \sum_i (w_i - \bar{w})^2\). Therefore,
\[
E\left(\frac{\delta \phi}{\phi}\right) \geq \frac{1}{16d} \sum_{(i,j) \in E} (w_i - w_j)^2 \\
= \frac{1}{16d} \sum_{(i,j) \in E} ((w_i - \bar{w}) - (w_j - \bar{w}))^2 \\
= \frac{1}{16d} \sum_{(i,j) \in E} (w_i - \bar{w})^2.
\]
Define \(x\) to be a vector of length \(n\) with elements \(x_j = w_j - \bar{w}\). Substituting,
\[
E\left(\frac{\delta \phi}{\phi}\right) \geq \frac{1}{16d} \sum_{(i,j) \in E} (x_i - x_j)^2 \\
= \frac{1}{16d} \left(\sum_{i,j \in E} (x_i - x_j)^2 \right) \left(\sum_{i=1}^n x_i = 0, x \neq 0\right)
\]
Since \(\sum_{i,j \in E} (x_i - x_j)^2 = x^T L x\) and \(\sum_{i=1}^n x_i^2 = x^T x\), this gives us
\[
E\left(\frac{\delta \phi}{\phi}\right) \geq \frac{1}{16d} \left(\frac{x^T L x}{x^T x} \right) \left(\sum_{i=1}^n x_i = 0, x \neq 0\right)
\]
Since $\sum x_i = 0$, $x$ is orthogonal to the eigenvector of the Laplacian matrix corresponding to $\lambda_1$, namely, $u = (1, 1, ..., 1)^T$. From Fact 2 in Section 2.1 it follows that

$$E\left(\frac{d\phi}{\phi}\right) \geq \frac{1}{16d} \max_{\Delta} \left(\frac{x^T L x}{x^T x}\right) x \perp u, x \neq 0 = \frac{\lambda_2}{16d}.$$  

It is worth noting that although load is moved only along a subset of edges forming a matching the convergence is in terms of the global properties of the graph, namely, $\lambda_2$ and $d$. Note that for any connected graph, $0 < (\lambda_2/2d) \leq 1$ [MP92]. Thus, Algorithm LR guarantees a positive fractional (possibly non-constant) decrease in the potential for any connected graph. Table I lists the values of $\lambda_2$ and $16d/\lambda_2$ for some graphs of typical interest.

<table>
<thead>
<tr>
<th>Processor Graph</th>
<th>$\lambda_2$</th>
<th>$16d/\lambda_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear array</td>
<td>$O(1/n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Star</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>2-dim mesh</td>
<td>$O(1/n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$d$-dim mesh</td>
<td>$O(1/n^{d/2})$</td>
<td>$O(n^{1/d})$</td>
</tr>
<tr>
<td>Hypercube</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Clique</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

**Lemma 3.** For a linear array there exists a load distribution for which no randomized algorithm on our model for the PLS problem can have a convergence ratio $\Omega(\lambda_2/2d)$.

**Proof.** Consider the load distribution $(n, 2n, 3n, ..., n^2)$ on the nodes $\{1, 2, ..., n\}$ respectively of the linear array. The average load $w = n(n+1)/2$. Then, the initial potential $\phi_0 = n^2((n^2/12) - (n/12)) \approx n^2/24$. It is clear that over the set of all possible matchings $M$,

$$E(\Delta \phi_M) \leq \max_{M} (\Delta \phi_M)$$

where $\Delta \phi_M$ denotes the decrease in $\phi$ due to a matching $M$ chosen by a randomized algorithm for PLS. But in any matching $M$, there can be at most $n/2$ edges and load movement across any edge $(k, k+1)$ can decrease the potential by at most $(1/2)(nk - (nk + n))^2 \approx n^2/24$. Whatever be the probability distribution imposed by the randomized algorithm on the matchings,

$$E\left(\frac{d\phi}{\phi}\right) \leq \frac{(n^2/2)(n/2)}{n^2/24} = \frac{6}{n^2}$$

For a linear array, $\lambda_2 \sim \pi^2/n^2$. Therefore, $\lambda_2/2d \sim \pi^2/4n^2$. Clearly, $E(\Delta \phi_M) \leq c(\lambda_2/2d)$, where $c = 24/\pi^2$. The lemma follows.  

**Lemma 4.** For a $d$-dimensional hypercube there exists a load distribution for which no randomized algorithm on our model for the PLS problem can have a convergence ratio $\Omega(\lambda_2/2d)$.

**Proof.** Assign $d$-bit addresses to the nodes of the hypercube according to the natural $d$-bit addressing scheme. Let $k$ denote the node which has all zeros in its address. Consider the load distribution $\mathbf{w}$ where the load on node $i$ is $w_i = 2H_{i,k}$, where $H_{i,k}$ is the Hamming distance between the addresses of nodes $i$ and $k$.

We use two properties of $\mathbf{w}$. First, for every edge $(i, j) \in E$, $|w_i - w_j| = 2$. This is because $|H_{i,k} - H_{j,k}| = 1$ when $(i, j) \in E$. Second,

$$\sum_{i \in V} (w_i - \bar{w})^2 = \sum_{j = 0}^{d} (d - 2j)^2 \left(\begin{array}{c} d \\
 \end{array}\right) = d2^d$$

This is easy to show by algebraic manipulation.

Using the first property of $\mathbf{w}$ and the fact that at most $1/d$ of the total edges in $E$ can be in any matching $M$, it follows that for any matching $M$, the potential drop due to load movement across the edges of $M$ is at most

$$\frac{1}{2} \sum_{(i,j) \in M} (w_i - w_j)^2 \leq \frac{1}{2d} \sum_{(i,j) \in E} (w_i - w_j)^2$$

Following arguments as in Lemma 3, for any matching $M$ chosen by a randomized algorithm for the PLS problem

$$E\left(\frac{d\phi}{\phi}\right) \leq \max_{M} (\Delta \phi_M) \leq \frac{1}{2d} \sum_{i \in V} (w_i - w_j)^2$$

Since for a hypercube $\lambda_2/2d = 1/d$, the lemma follows.
Lemma 5. For a clique there exists a load distribution for which no randomized algorithm on our model for the PLS problem can have a convergence ratio $\Omega(\lambda_2/2d)$.

Proof. Consider the load distribution $w = (0, n, n, \ldots, n)$ on the nodes $\{1, 2, \ldots, n\}$ of the clique respectively. Let $E_i$ denote the set of edges incident on node $i$. Note that for $(i, j) \in E_i$, $|w_i - w_j| = n$ and for $(i, j) \in E - E_i$, $|w_i - w_j| = 0$.

Any algorithm for the PLS problem (even one which is randomized and has global information) has to choose one and at most one edge (say, $(p, q)$) from $E_i$ to cause any reduction in $\phi$. Therefore, the potential drop $\delta \phi_{i,j}$ due to load movement across the edges in $M$ can be at most $1/(2(n - 1)) \sum_{(i, j) \in E} (w_i - w_j)^2 = (1/(2(n - 1))) \sum_{i \in V} (w_i - w_i)^2$. Following arguments as in Lemma 3 and 4,

$$E\left(\frac{\delta \phi_{i,j}}{\phi}\right) \leq \frac{1}{2(n-1)} \frac{\sum_{i \in V} (w_i - w_i)^2}{\sum_{i \in V} w_i} = \frac{1}{2(n-1)} \frac{n}{n^2 - n} = \frac{n}{2(n-1)}$$

Since $\lambda_2 = n$ for a clique, $\lambda_2/2d = n/(2(n-1))$. The lemma follows.

3.1.2. Algorithm LR with Integral Weights. We now extend our result from Section 3.1.1 to the case when the loads are integral. Note that in this case, the loads at the endpoints of an edge cannot be equalized beyond a precision of one unit. We modify Algorithm LR and obtain Algorithm Discrete-Local-Random (DLR).

In Algorithm LR a matching $M$ is chosen locally in the same manner as in Algorithm LR. The only difference is in load equalization for the edges in $M$. Assume that an edge $(i, j)$ has been chosen in $M$ and without loss of generality $w_i \geq w_j$. When $(w_i, w_j)$ is even, load equalization as in Algorithm LR suffices. But when $(w_i, w_j)$ is odd, a total of $w_i - ((w_i + w_j - 1)/2)$ load units are moved from node $i$ to node $j$. Note that the new loads on nodes $i$ and $j$ after this transfer are $(w_i + w_j + 1)/2$ and $(w_i + w_j - 1)/2$ respectively.

Theorem 2. Applying Algorithm DLR on any connected graph $G$ with integral load distribution $w$ and potential $\phi$, $E(\delta \phi) \geq \frac{\lambda_2}{16d(1+\varepsilon)} \frac{\phi}{d n/\lambda_2}$ if $\phi \geq \left(1 + \frac{1}{\varepsilon}\right) d n/\lambda_2$.

Proof. Let $E_i \subseteq E$ and $E_e \subseteq E$ denote the sets of edges for which $(w_i, w_j)$ is even and is odd respectively. Let $\delta \phi_{i,j}$ denote the decrease in potential due to movement of load across edge $(i,j)$. Note that,$$
\delta \phi_{i,j} = \begin{cases} \frac{(w_i - w_j)^2}{2} & \text{if } (i,j) \in E_e \\ \frac{(w_i - w_j)^2 - 1}{2} & \text{if } (i,j) \in E_e \end{cases}$$

Now, $E(\delta \phi) = \sum_{(i,j) \in E_e} E(\delta \phi_{i,j}) + \sum_{(i,j) \in E_e} E(\delta \phi_{i,j})$. Using Lemma 2,

$$E(\delta \phi) \geq \frac{1}{16d} \sum_{(i,j) \in E_e} (w_i - w_j)^2$$

$$+ \frac{1}{16d} \sum_{(i,j) \in E_e} ((w_i - w_j)^2 - 1).$$

Since $E_e \cup E_e = E$,

$$E(\delta \phi) \geq \frac{1}{16d} \sum_{(i,j) \in E_e} ((w_i - w_j)^2 - 1)$$

$$= \frac{1}{16d} \sum_{i \in V} (w_i - w_i)^2 - e/\phi,$$

where $e \leq nd$ is the number of edges in $G$.

As in Theorem 1, the first term above is at least $\lambda_2$. Therefore,

$$E(\delta \phi) \geq \lambda_2 - \frac{n}{16d} - \frac{\phi}{16d}.$$ 

Clearly when $\phi \geq (1 + (1/\varepsilon)) d n/\lambda_2$, we have $E(\delta \phi) \geq \lambda_2/16d(1+\varepsilon).$ This proves the first claim in the theorem.

Now consider the case when $\phi < (1 + (1/\varepsilon)) d n/\lambda_2$. From inequality (2), as long as there exists an edge $(i,j)$ such that $|w_i - w_j| \geq 2$, $\delta \phi \geq 3/16d$. Note that if for every edge $(i,j) \in E$, $|w_i - w_j| \leq 1$, then the load-balanced state has already been reached.

It should be noted that the potential requirement $(1 + (1/\varepsilon)) d n/\lambda_2$ in Theorem 2 above which $\phi$ provably decreases by a multiplicative factor and below which it provably decreases by at least an additive term depends only on the structure and size of the given graph $G$ and not on the initial load distribution on the nodes.

Remark. The optimality claims in Subsection 3.1.1 concerning the lower bound on the convergence ratio apply here as well. That is, Algorithm DLR has an asymptotically optimal convergence ratio when $\phi \geq (1 + (1/\varepsilon)) d n/\lambda_2$. When the potential is small, the case of integral load units is intrinsically harder than the real load case as the following theorem shows.
**Theorem 3.** For any algorithm for the PLS problem with integrals loads, there exists an input graph and load distribution \( w \) such that when \( \phi < (1 + (1/\varepsilon)) \frac{dn}{\lambda_2} \), where \( 0 < \varepsilon \leq 1 \), the convergence ratio for the algorithm is \( o(\lambda_2/2d) \).

**Proof.** Consider a linear array with the load distribution \( w = (1, 2, \ldots, n-1, n+1) \) on the nodes \( \{1, 2, \ldots, n\} \) respectively.

For a linear array, \( \lambda_2 \approx \pi^2/n^2 \). Therefore, \( \frac{dn}{\lambda_2} \approx 2n^2/n^2 \). The average load \( \bar{w} = (n+1)+1/n \). Therefore \( n/12 \leq \phi \leq n^3/6 \). Choosing \( \varepsilon = 1 \), \( \phi \leq 2dn/\lambda_2 \).

Any algorithm for the PLS problem with integrals loads can reduce the potential only by moving load across edge \((n-1, n) \). The maximum drop in potential is

\[
\delta \phi \leq \frac{1}{2}((n+1)-(n-1))^2 = 2.
\]

Therefore the maximum convergence ratio for any algorithm is

\[
\frac{\delta \phi}{\phi} \leq \frac{2}{n^2/12} = \frac{24}{n^2} = o\left(\frac{1}{n^2}\right) = o\left(\frac{\lambda_2}{2d}\right).
\]

This proves the theorem. \( \square \)

**3.2. PLS with Edge Failure**

Now we consider the case when edges possibly fail while an algorithm for the PLS problem is being executed. Recall from Section 2.2 that edges fail only between time steps and when an edge fails, the processors at its endpoints know about it immediately. We modify the implementation of Algorithm DLR (Fig. 1) in a simple manner to accomodate failing edges. In Step \( \mathcal{A} \) where a matching is chosen, if edge \((i, j)\) fails between any two time steps and \(j\) is the partner of \(i\), then processor \(i\) is marked \(W\). In Step \( \mathcal{B}\), if edge \((i, j)\) fails before the load movement begins, then no load is moved along \((i, j)\). Recall that a communication or load movement once begun on an edge is guaranteed to be successful.

**Theorem 4.** For any connected graph \(G\) and load vector \(w\), the modified DLR Algorithm produces a decrease \(\delta \phi\) in potential such that

\[
E(\delta \phi) \geq \begin{cases} 
\frac{\lambda_2(G)}{16(1+\varepsilon)d(G)} \phi & \text{if } \phi \geq (1+\frac{1}{\varepsilon}) \frac{dn}{\lambda_2} \\
\frac{3}{16d(G)} & \text{if } \phi < (1+\frac{1}{\varepsilon}) \frac{dn}{\lambda_2}
\end{cases}
\]

where \(0 < \varepsilon \leq 1\), \(d(G)\) is the degree of \(G\), \(H\) is the subgraph of \(G\) obtained by removing the failed edges at the end of the algorithm and \(\lambda_2(H)\) is the second smallest Laplacian eigenvalue of the Laplacian of \(H\).

**Proof.** Our argument is similar to the backward analysis in [S93]. We claim that for each edge \((i, j) \in H\),

\[
\Pr[(i, j) \in M \text{ at the end of Step } \mathcal{A}'] \geq 1/8d(G) \cdot (\delta \phi)
\]

This claim follows from the proof of Lemma 2 by restricting the proof to edges \((i, j) \in H\) rather than edges \((i, j) \in G\).

Recall that in Lemma 2 we have proved that for \((i, j) \in G\), the probability that \((i, j)\) is picked to be in \(M\) is at least \(1/8d(G)\). We note the following intuition. In Algorithm modified DLR, the links in \(S\) are chosen just as in Fig. 1. They can either be isolated links or connected components consisting of more than one link. Without link failure, Algorithm DLR picks only the isolated edges in \(S\) to be in the final matching \(M\). If any link \((i, j) \in H\) turns out to be an isolated link in \(S\), then it is picked by the modified DLR algorithm to be in \(M\) just as before, irrespective of the failure of neighboring links. On the other hand, if \((i, j) \in H\) is not an isolated link in \(S\), then it might be picked to be in \(M\) if all its neighboring links in \(S\) fail (in contrast, without link failure, such a link would not have been picked to be in \(M\)). Therefore, the probability that a link \((i, j) \in H\) is picked to be in \(M\) does not decrease under possible link failure. Hence Inequality 3 still holds.

The rest of the argument is as in Theorem 2. \( \square \)

**4. Dynamic Load Balancing**

In this section we use our algorithm for the PLS problem with integral weights to obtain an efficient algorithm for the dynamic load balancing problem. The dynamic load balancing problem can be solved by repeatedly invoking Algorithm DLR and moving incrementally towards the load-balanced state. We first consider the case when links do not fail. We then extend our solution to the case when links possibly fail.

**4.1. Dynamic Load Balancing without Link Failures**

**Theorem 5.** Given a graph with \(n\) nodes, maximum degree \(d\) and an arbitrary initial load distribution, the dynamic load balancing problem can be solved by invoking Algorithm DLR

\[
O\left(\frac{d\log \phi_0}{\lambda_2} + \frac{d^2n}{\lambda_2}\right)
\]

times with high probability, where \(\phi_0\) is the initial potential and \(\lambda_2\) is the second smallest Laplacian eigenvalue of the graph.

**Proof.** Let \(\phi_k\) be the random variable denoting the potential of processor graph \(G\) after the \(k\)th invocation of Algorithm DLR. By Theorem 2, as long as \(\phi_k = \Omega(\frac{dn}{\lambda_2})\), in
the \((k + 1)\)th invocation of the Algorithm DLR the expected decrease in the potential is at least a factor of \(\gamma = \Omega(\log(1/\delta_2))\). Therefore the expected number of times Algorithm DLR is invoked before the potential becomes \(O(\log(1/\delta_2))\) is \(O(\log(1/(1 - \log(1/\delta_2))))\). After this happens, by Theorem 2, on each invocation of DLR the expected decrease in the potential is by an additive \(3\log(1/\delta_2)\). Thus with high probability, the number of times Algorithm DLR is invoked is \(O(k^*(\log(\phi_0) + \log n)) = O((d\log(\phi_0)/\delta_2) + (d\log(\phi_0)/\delta_2))\), proving the theorem.

Remark 1. For intuition, we make several comments comparing this result to that in \([C89]\) on the multiport model. A step of diffusive load balancing algorithms, such as ours or similar ones on the multiport model, can be modeled as follows:

\[
w' = Mw
\]

where \(w\) and \(w'\) are the weight vectors at the beginning and the end of the step, respectively. Here \(M\) is a doubly stochastic symmetric matrix (for technical reasons, see \([C89]\)). From the results in \([C89]\), we can conclude that repeated application of such steps roughly \(O(\log(\phi_0/(1 - \beta^2)))\) times yields an algorithm for load-balancing; here, \(\beta\) is the second largest eigenvalue of \(M\) in magnitude\(^1\). In what follows, we compare this result with ours.

Comment 1. This result does not give a non-trivial analysis of our algorithm on the matching model. To see this, note that Algorithm LR can be expressed as follows:

\[
w_j = (I - \frac{1}{2}L_M)w_j
\]

where \(w_j\) and \(w'\) are the initial and final weight vectors, \(I\) is an \(n\) by \(n\) identity matrix, \(M\) is set of edges in the matching chosen by Algorithm LR and \(L_M\) is the Laplacian matrix of the graph \(G' = (V, M)\). Note that the graph \(G'\) is a subgraph of \(G\) consisting of the edges in \(M\). Therefore \(G'\) consists of isolated edges and possibly isolated nodes. The distinct eigenvalues of \(L_M\) turn out to be 0 and 2. So, the second largest eigenvalue of \(I - L_M\) in magnitude, namely \(\beta\), is precisely 1. That gives the trivial bound of infinity on the number of times Algorithm LR is invoked to balance load. Thus, the approach in \([C89]\) does not give a non-trivial analysis for our algorithm.

Comment 2. Given some preprocessing we can directly utilize this result on the matching model. For example, by using a preprocessing phase of edge-coloring \(G\), we can simulate each step \((*)\) on the matching model in \(O(d)\) steps (since a graph of degree \(d\) can be edge-colored using \(O(d)\) colors and all edges corresponding to any color form a matching). Besides adding a \(O(d)\) factor to the number of steps for load balancing, such a simulation seems hard to analyze when there are link failures.

How does the bound for load balancing achieved by this simulation compare with our bound in Theorem 5? We are not able to answer that question in general for an arbitrary \(M\).

\(^1\) The analysis in \([C89]\) does not directly extend to the case when the weights are integral. Recently, the analysis there has been extended to handle the integral weights as well \([GMS95]\).
However, for a number of ways of choosing $M$ locally by the processors, we are able to show that the bound in Theorem 5 is better. In what follows we give one such example of $M$.

**Example.** Consider $M$ in which $M_{ij} = 1/2d$ for $(i,j) \in E$ (and therefore, $i \neq j$). We can show $\beta = 1 - (\lambda_2/2d)$ for that $M$. We have $1 - \beta^2 = (\lambda_2/d) - (\lambda_2^2/4d^2)$. Thus, $1 - \beta^2 = \Theta(\lambda_2/d)$. Then the result above shows that $O(d \log \phi_0/\lambda_2)$ steps are needed to do load balancing on the multiprocessor model. Simulating that on the multiprocessor model takes $O(d^2 \log \phi_0/\lambda_2)$ steps as remarked above. In contrast, from Theorem 5 it follows that our algorithm takes only $O(d \log \phi_0/\lambda_2)$ steps for sufficiently large $\phi_0$.

**Remark 2.** The bound in Theorem 5 holds for an arbitrary graph. This compares poorly with the bounds known for some specific graphs. For example, on 2-dim meshes and hypercubes, load balancing can be done (using [C89]) in $O(D)$ steps on the matching model where $D$ is the diameter of the graph. That is, the number of steps taken is $O(\sqrt{n})$ and $O(\log n)$ respectively for the 2-dim mesh and the hypercube; in contrast, our algorithm takes $O(n \log \phi_0)$ and $O(\log n \log \phi_0)$ steps respectively from Theorem 5. The strength of our algorithm and analysis is that it uniformly applies to every graph.

4.2. Dynamic Load Balancing with Link Failures

Here we use the modified Algorithm DLR from Section 3.2.

**Theorem 6.** Given a processor network with initial topology $G$, load assignment $w$ and initial potential $\phi_0$, dynamic load balancing can be solved by invoking Algorithm DLR

$$O \left( \frac{d(G) \log \phi_0}{\lambda_2(H)} + \frac{d(G)^2 n}{\lambda_2(H)} \right)$$

times with high probability, where $d(G)$ is the degree of $G$, $H$ is a graph representing the final network without considering the failed links $\lambda_2(H)$ is the second smallest eigenvalue of the Laplacian of $H$.

**Proof.** The argument is as in the proof of Theorem 5 except that Theorem 4 is used instead of Theorem 2. 

4.3. Extension to Domain Repartitioning Problems

Consider parallel applications in mechanical engineering and visualization software which use locally adaptive finite-element or finite-difference meshes to solve partial differential equations (PDEs). In these applications, a PDE is numerically solved on some data domain which is discretized using a mesh of finite elements or points. For parallel solution, this domain is divided into subdomains and each subdomain is mapped to a processor. In adaptive mesh terminology, the graph representing the subdomain connectivity information is called the quotient graph. Each node (subdomain) in the quotient graph represents a number of mesh points or elements.

Due to computations within each processor the elements in subdomain are dynamically coarsened (i.e., mesh points or elements are coalesced or deleted) or refined (i.e. mesh points or elements are subdivided or added on). This causes a load imbalance between the processors and repartitioning of the domain therefore becomes necessary to achieve balanced load. Achieving balanced subdomains usually involves shifting the boundaries of adjoining subdomains so as to equalize the mesh points or elements in each subdomain. Further references on these areas can be found in [BB87, HT93, W91].

Clearly, our algorithm for the PLS problem can be used repeatedly on the quotient graph to solve the load balancing problem in adaptive mesh partitioning. Each load unit is a point or element in the mesh. The actual migration of load units as determined by the application of our algorithm can be performed on the underlying architecture by either local communication (if adjoining subdomains have been mapped to neighboring processors) or by non-local routing (if adjoining subdomains have been mapped to non-neighboring processors).

As such, adaptive mesh partitioning involves optimizing a number of parameters (e.g., minimization the size of the boundaries of subdomains by moving appropriate sets of mesh points etc) besides merely balancing the load. Our solution does not address the optimization of these other parameters. For example, although our solution indicates how many mesh points must be moved between the nodes, it does not determine which mesh points must be moved. However iterative algorithms such as the one we suggest are frequently used in adaptive mesh partitioning in practice (for example, see [HT93, WB92, WCE95]). For a comparison of the iterative schemes with other schemes such as recursive bisection, spectral methods and simulated annealing in adaptive mesh partitioning, see [OD + 93, WB92, W91] and references therein.

**5. Experimental Observations**

An assessment of repeated invocations of Algorithm LR and Algorithm DLR for the problem of dynamic load balancing was obtained through simulation and experimentation on processor graphs of different sizes and connectivities. We discuss two important issues and present a small sample of our experimental data. In most of our experiments, the load on each processor was chosen uniformly and randomly from the interval $(0, a)$ for various
values of \(a\). In what follows, such a load distribution is denoted by \(\text{Random}(0, a)\).

1. **Real versus Integral Loads.** The simplicity of our analysis is based on the fact that as long as the potential is large, the case when the loads are integral is very similar to the case when the loads are real. More precisely, we showed in Section 3.1.2 that the convergence ratio of Algorithm of DLR is at least half that of the theoretical lower bound of \(\frac{\lambda_2/2}{16d}\) on the convergence ratio for Algorithm LR as long as the potential is larger than \((1 + (1/\varepsilon)) d\lambda_2\). Let this cutoff potential be denoted by \(\phi_c\). The theoretically predicted value of \(\phi_c\) is \(2d\lambda_2\) (setting \(\varepsilon = 1\)).

The following are the results of our experiments to study how closely the convergence ratios of the Algorithm DLR and Algorithm LR behave. Figures 3 and 4 show the decrease in potential (averaged over 20 runs) for 80 invocations of Algorithm LR and Algorithm DLR on two graphs of 64 nodes each. In all our experiments, we observed that convergence ratios for real and integral loads were very similar even when the potential is considerably smaller than the predicted value of \(\phi_c\).

Consider Fig. 3. The theoretically predicted cutoff point in the plot is \(\log(2d\lambda_2/\lambda_2) \approx 7.5\). However, consider the plot between the 60th and 70th invocations. The initial potential at the 60th invocation of Algorithm LR is \(e^3\) and \(e^{3.5}\) respectively. After the 70th invocation the potential is approximately \(e^2\) and \(e^3\) respectively. Therefore the observed average convergence ratio between the 60th and 70th invocations for Algorithm LR is \(1 - e^{(3-3.5)/10} \approx 0.095\) whereas for Algorithm DLR it is \(1 - e^{(3-3.5)/10} \approx 0.048\). So, the convergence ratio for DLR becomes approximately half that of Algorithm LR only after the potential becomes \(e^{3.5}\) which is much less than the theoretically predicted potential of \(e^{7.5}\).

Our experiments did not indicate that there was a precise cutoff potential for any given graph. This is because the observed value of \(\phi_c\) differs for different initial load distributions. For example, Figures 4 and 5 shows that the observed values of \(\phi_c\) on a 64-node hypercube are different (\(e^{4.5}\) and \(e^{3.5}\), respectively) for two different initial load distributions. We could not obtain a concrete relationship between \(\phi_c\) and other parameters such as initial load distribution, special graph topologies, etc. from our experiments. We
believe that extensive studies are needed to ascertain such relationships and we leave that open.

2. Predicted versus Observed Convergence Ratio. Recall that our main results provide only a lower bound on the convergence ratios of our algorithms. The observed average convergence ratio of Algorithm LR was consistently greater than the predicted bound across all load distributions and graphs that were considered. This is not surprising since the lower bound on the convergence ratio can be attained only for a very restricted class of load assignments to the processors.

Define $R$ to be the ratio of the experimentally observed convergence ratio of Algorithm LR to its theoretical lower bound of $\lambda_1/16d$. Figure 6 shows the plot of $R$ (averaged over 20 runs of LR) versus the number of invocations of LR. Observe that the average of the ratio $R$ decreases with the increasing edge-density of the graph. That is, the observed convergence ratio is significantly more than the theoretical lower bound for sparse graphs and are comparable in the case of dense graphs. In fact the experimentally observed values of $R$ on sparser graphs for the same initial load distribution as in Figure 6 were much higher than those for the hypercube and the random graph; for example, 0 to 1000 for a linear array and between 0 and 80 for a 2-dim mesh (they could not be fitted into the scale of Fig. 6).

We experimented further with different initial load distributions (eg., a spike of $x$ units on a randomly chosen node and 0’s on the others) and graphs of varying connectivity (eg., a random sparse graphs, random dense graphs, standard graphs such as hypercubes and meshes etc.). The general trend in the results we observed were similar to above and therefore we do not include them here. A better evaluation of our algorithms can be obtained from analyzing real applications on parallel and distributed machines.

6. DISCUSSION

Our approach here can be applied to the model in which only one unit of load may be moved along any edge in a step. Appropriately applying the approach in this paper, we can analyze a simple local algorithm in that scenario as well. However we do not include the details of that here since an improved, in fact, an optimal analysis of that algorithm has been recently given [GL+95] using a substantially different approach.

We have proved that our algorithm and the analysis for the PLS problem is optimal in the matching model. We repeatedly applied that algorithm to obtain one for load balancing: is the resultant algorithm for load balancing optimal in the matching model? We leave that question open.

ACKNOWLEDGEMENTS

The authors thank Ravi Boppana, Stan Eisenstat, László Lovász, Eric Mjolsness, and Martin Schultz for their critical feedback and encouragement. They also thank Nick Carriero and Rob Bjornson for discussions about the communication hardware of various modern parallel computers. Finally, the authors sincerely thank the referees for their thorough and insightful comments.

REFERENCES


![Plot of ratio R](image_url)