On Scheduling Collaborative Computations on the Internet, I:
Mesh-Dags and Their Close Relatives*

(Extended Abstract)

Arnold L. Rosenberg
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003, USA
rsnbrg@cs.umass.edu

Abstract

Advancing technology has rendered the Internet a viable medium for collaborative computing, via mechanisms such as Web-Based Computing and Grid-Computing. We present a “pebble game” that abstracts the process of scheduling a computation-dag for computing over the Internet, including a novel formal criterion for comparing the qualities of competing schedules. Within this formal setting, we identify a strategy for scheduling the task-nodes of a computation-dag whose dependencies have the structure of a mesh of any finite dimensionality (a mesh-dag), that is optimal to within a small constant factor (to within a low-order additive term for 2- and 3-dimensional mesh-dags). We show that this strategy remains nearly optimal for a generalization of 2-dimensional mesh-dags whose structures are determined by abelian monoids (a monoid-based version of Cayley graphs).

1. Introduction

Advancing technology has rendered the Internet a viable medium for collaborative computing, via mechanisms such as Web-Based Computing (WBC, for short) and Grid-Computing (GC, for short). Both WBC and GC depend on remotely situated “volunteers” to collaborate in the computation of a massive collection of (usually compute-intensive) tasks. A WBC project proceeds essentially as follows. Interested volunteers register with a WBC website. Thereafter, each registered volunteer visits the website from time to time to receive a task to compute. Some time after completing the task, the volunteer returns the results from that task and receives a new task. And the cycle continues. Interesting examples of WBC projects are [13, 22], which benefit from Internet computing because of the sheer volume of their workloads, and [12, 14, 21], which benefit because of the computational complexity of their individual tasks. A Computational Grid is a consortium of (usually geographically dispersed) computing sites that contract to share resources; see [3, 4, 7, 8]. In contrast to the “pull”-based allocation of tasks in a WBC project, GC projects often have Grid members “push” tasks to other members. Such differences aside, the collaboration inherent in both WBC and GC enables a large range of computations that cannot be handled efficiently by any fixed-size assemblage of dedicated computing agents (e.g., a multiprocessor or a NOW). Given the initial successes of Internet computing—see the cited sources—one can envision the Internet’s becoming the computing platform of choice for a variety of computational problems that are prohibitively consumptive of resources on traditional computing platforms.

As with every major technological advance in collaborative computing, WBC and GC create significant new scheduling challenges even while enabling new levels of computational efficiency. In the case of WBC and GC, a major challenge arises from the fact that both WBC volunteers and participating members of a Grid promise to return the results of executing an allocated task only eventually. This lack of any time guarantee creates a nontrivial scheduling challenge when the tasks comprising the collaborative workload have interdependencies that constrain their order of execution. Specifically, such dependencies can potentially engender gridlock when no new tasks can be allocated for an indeterminate period, pending the execution of already allocated tasks. Although “safety devices” such as deadline-triggered reallocation of tasks address this danger, they do not eliminate it, since the “backup” WBC volunteer or Grid member assigned a given task may be as dilatory as the primary one.

* Supported in part by NSF Grant CCR-00-73401.
In this paper, we study this scheduling problem for the common situation in which the collaborative workload’s intertask dependencies are structured as a dag; cf. [9, 10]. Specifically, we study how to schedule the allocation of tasks of a computation-dag in a way that minimizes the danger of gridlock, by provably (approximately) maximizing the number of tasks that are eligible for execution at each step of the computation.

The contributions of our study are of three types. (1) We develop a variant of the pebble games that have proven so useful in the formal study of a large variety of scheduling problems, that is tailored for Internet-computing. A major facet of devising such a computing model is developing a formal criterion for comparing the quality of competing schedules. We formulate such a criterion for computation-dags whose dependencies have the structure of a mesh of some finite dimensionality (a mesh-dag, for short). The quality-measuring component of our model formalizes and quantifies the intuitive goal that a schedule “maximizes the number of tasks that are eligible for execution at each step of the computation.” (2) We identify a strategy for scheduling mesh-dags, that is provably optimal (according to our formal criterion) to within a small constant factor (to within a low-order additive term for 2- and 3-dimensional mesh-dags). This strategy allocates task-nodes of a mesh-dag $M$ along $M$’s successive “diagonal” levels. (3) We augment the second contribution by showing that this level-by-level scheduling strategy remains nearly optimal for a generalization of 2-dimensional mesh-dags whose structures are determined by abelian monoids (a monoid-based version of Cayley graphs).

2. A Formal Model for Internet Computing

In this section, we develop the entities that underlie our study: computation-dags and the pebble games that model the process of executing the dags’ computations.

2.1. Computation-dags

A directed graph (digraph, for short) $G$ is given by: a set of nodes $N_G$ and a set of arcs (or, directed edges) $A_G$, each having the form $(u \rightarrow v)$, where $u, v \in N_G$. A path in $G$ is a sequence of arcs that share adjacent endpoints, as in the following path from node $u_1$ to node $u_n$:

$$(u_1 \rightarrow u_2), (u_2 \rightarrow u_3), \ldots, (u_{n-2} \rightarrow u_{n-1}), (u_{n-1} \rightarrow u_n)$$

A dag (directed acyclic graph) $G$ is a digraph that has no cycles; i.e., in a dag, the preceding path cannot have $u_1 = u_n$. When a dag $G$ is used to model a computation (is a computation-dag): (a) each node $v \in N_G$ represents a task; (b) an arc $(u \rightarrow v) \in A_G$ represents the dependence of task $v$ on task $u$: $v$ cannot be executed until $u$ is. For each $(u \rightarrow v) \in A_G$, we call $u$ a parent of $v$ and $v$ a child of $u$.

We purposely do not posit the finiteness of computation-dags. While the intertask dependencies in nontrivial computational jobs usually have cycles—typically caused by loops—it is useful to “unroll” these loops when scheduling the individual tasks in such jobs. This converts the job’s (often modest-size) computation-digraph into an evolving set of expanding “prefixes” of what eventually becomes an enormous computation-dag. One often has better algorithmic control over the “steady-state” scheduling of such jobs if one expands these computation-dags to their infinite limits and concentrates on scheduling tasks in a way that leads to a computationally expedient set of evolving prefixes. In this paper, we study an important class of such jobs: the large variety of (say, linear-algebraic) computations involving often-enormous matrices, whose associated computation-dags are mesh-dags.

2.2. Mesh-structured computation-dags

Let $N$ denote the set of nonnegative integers. For each positive integer $d$, the $d$-dimensional mesh-dag $M_d$ has node-set $N^d$, the set of $d$-tuples of nonnegative integers. The arcs of $M_d$ connect each node $(v_1, v_2, \ldots, v_d) \in N^d$ to its $d$ children, $(v_1, v_2, \ldots, v_j + 1, \ldots, v_d)$, for all $1 \leq j \leq d$. Node $(0, 0, \ldots, 0)$ is $M_d$’s unique parentless node. The diagonal levels of the dags $M_d$ play an essential role in our study. Each such level is the subset of $M_d$’s nodes that share the sum of their coordinates: each level $\ell = 0, 1, \ldots$ of $M_d$ is the set $L^{(d)}_\ell \equiv \{(v_1, \ldots, v_d) \mid v_1 + \cdots + v_d = \ell\}$. See Fig. 1.

![Figure 1. The first four diagonal levels of $M_2$.](image)
a diverse range of problems related to scheduling the tasknodes of a computation-dag \( G \) [5, 11, 15]. Such games use tokens (called pebbles) to model the progress of a computation on \( G \); the placement or removal of pebbles of various types—which is constrained by the dependencies modeled by \( G \)'s arcs—represents the changing (computational) status of the tasks represented by \( G \)'s nodes. The pebble game that we study here is essentially the “no recomputation allowed” pebble game from [20], but it differs from that game in the resource one strives to optimize. For brevity, we describe only the WBC version of the game here; the simpler GC version should be apparent to the reader.

The rules of the game. The Web-Oriented (W-O, for short) Pebble Game on a dag \( G \) involves one player \( S \), the Server, and an indeterminate number of players \( C_1, C_2, \ldots, \) the Clients. The Server has access to unlimited supplies of three types of pebbles: ELIGIBLE-BUT-UNALLOCATED (EBU, for short) pebbles, ELIGIBLE-AND-ALLOCATED (EAA, for short) pebbles, and EXECUTED pebbles. The Game’s rules are described in Fig. 2.

The W-O Pebble Game’s moves reflect the successive stages in the “life-cycle” of a node in a computation-dag, from eligibility for execution through actual execution. The moves also reflect the danger of a game’s being stalled indefinitely by dilatory Clients (the “gridlock” of Section 1).

The quality of a play of the game. Our goal is to determine how to play the W-O Pebble Game in a way that maximizes the number of EBU nodes at every moment—so that we maximize the chance that the Server has a task to allocate when approached by a Client. We believe that no single formalization of this goal is appropriate to all computation-dags; therefore, we focus on specific classes of dags of similar structures—mesh-like dags in this paper, tree-dags in [18]—and craft a formal quality criterion that seems appropriate for each class. We hope that the analyses that uncover the optimal scheduling strategies for these specific classes will suggest avenues toward crafting criteria that are appropriate for other specific classes of computation-dags.

We begin our formulation of a quality criterion by ignoring the distinction between EBU and EAA pebbles, lumping both together henceforth as ELIGIBLE pebbles. We do this because the number \( n \) of ELIGIBLE pebbles at a step of the W-O Pebble Game is determined by the structure of the computation-dag in question and by the strategy we use when playing the Game. The breakdown of \( n \) into the relative numbers of EBU and EAA pebbles reflects the frequency of approaches by Clients during that play—which is beyond our control!

Given our focus on mesh-dags, we restrict attention to families \( G \) of computation-dags such that each \( G \in G \):

- is infinite: The set \( \mathbb{N} \) is infinite;
- has infinite width: For each integer \( n > 0 \), in every successful play of the Game on \( G \), there is a step at which \( \geq n \) nodes of \( G \) contain ELIGIBLE pebbles.

One final simplifying assumption will lead us to a tractable formalization of our informal quality criterion, that is suitable for mesh-like dags:

- Tasks are executed in the same order as they are allocated.

Within the preceding environment, we strive to maximize, at each step \( t \) of a play of the W-O Pebble Game on a computation-dag \( G \), the number of ELIGIBLE pebbles on \( G \)'s nodes at step \( t \), as a function of the number of EXECUTED pebbles on \( G \)'s nodes at step \( t \). In other words, we seek a schedule \( S \) for allocating ELIGIBLE tasks, that produces the slowest growing enabling function for \( G \):

\[
E_G(n; S) = \max \{ n \in \mathbb{N} : \text{ EXECUTED under } S \text{ before there are } \geq n \text{ ELIGIBLE pebbles on } G \}.
\]

In contrast to historical pebble games, which strive to assign EXECUTED pebbles as quickly as possible, we strive to allocate ELIGIBLE pebbles as quickly as possible.

3. Near-Optimal Schedules for Mesh-Dags

Theorem 1 (a) (i) For any schedule \( S \) for \( \mathcal{M}_2 \),

\[
E_{\mathcal{M}_2}(n; S) \geq \binom{n}{2}.
\]

(ii) Any schedule \( S \) for \( \mathcal{M}_2 \) that schedules nodes along the diagonal levels of \( \mathcal{M}_2 \) has

\[
E_{\mathcal{M}_2}(n; S) < \binom{n}{2} + n.
\]

(b) (i) For any schedule \( S \) for \( \mathcal{M}_d \),

\[
E_{\mathcal{M}_d}(n; S) = \Omega(n^{\frac{1}{d-1}}).
\]

(ii) Any schedule \( S \) for \( \mathcal{M}_d \) that schedules nodes along diagonal levels has

\[
E_{\mathcal{M}_d}(n; S) = \Theta(n^{\frac{1}{d-1}}).
\]

The constants “hiding” in the big-\( \Omega \) and the big-\( \Theta \) depend on \( d \) but not on \( n \).

Proof Sketch. Our analysis of schedules for the mesh-dags \( \mathcal{M}_d \) proceeds by induction on \( d \). Part (a) forms the basis of our induction, part (b) its extension.

(a)(i) Focus on a time \( t \) when \( n \) nodes of \( \mathcal{M}_2 \) are ELIGIBLE. Call the induced subdag of \( \mathcal{M}_2 \) on the set of nodes \( \{i\} \times \mathbb{N} \) (resp., \( \mathbb{N} \times \{j\} \) the \( i \)th row (resp., the \( j \)th column) of \( \mathcal{M}_2 \). The dependencies of \( \mathcal{M}_d \) guarantee that:

- No two ELIGIBLE nodes reside in the same row or the same column of \( \mathcal{M}_2 \).
- Every row- and column-ancestor (parent, parent’s parent, . . .) of each ELIGIBLE node must be EXECUTED by time \( t \).

We must, thus, have \( n \) distinct rows, each containing a distinct (perforce, nonnegative) number of EXECUTED nodes.
The number of **EXECUTED** nodes must, therefore, be no smaller than
\[ \sum_{k=0}^{n-1} k = \binom{n}{2}. \]

(a)(ii) Let \( S \) be any schedule for \( M_d \) that allocates **ELIGIBLE** nodes along successive diagonal levels. To see that \( S \) achieves the advertised upper bound, note that: (1) The \( n \) nodes of \( M_d \) that lie on level \( L_{n}^{(2)} \) are all **ELIGIBLE** as soon as all of their \( \binom{n}{2} \) ancestors have been **EXECUTED**.

(2) While proceeding from the position wherein the **ELIGIBLE** nodes comprise level \( L_{n}^{(2)} \) to the analogous position for level \( L_{n+1}^{(2)} \), the number of **EXECUTED** nodes always has the form \( \binom{n}{2} + k \) for some \( k < n \). See Fig. 3.

(b)(i) At any time \( t \) when \( n \) nodes of \( M_d \) are **ELIGIBLE** for execution, \( M_d \)'s dependencies guarantee that all ancestors of each **ELIGIBLE** node of \( M_d \) must be **EXECUTED**. Consequently:

### Fact 1

**For each index** \( a \in \{1, 2, \ldots, d\} \), **for each** \( \langle u_1, \ldots, u_{a-1}, u_a+1, \ldots, u_d \rangle \in \mathbb{N}^{d-1} \), **there is at most one** \( u_a \in \mathbb{N} \) **such that** \( \langle u_1, \ldots, u_{a-1}, u_a, u_{a+1}, \ldots, u_d \rangle \) **is an **ELIGIBLE** node of** \( M_d \).

We operate with the following inductive hypothesis.

**Inductive Hypothesis.** For each dimensionality \( \delta < d \), there exists a constant \( c_\delta > 0 \) such that: for any schedule \( S \) for \( M_\delta \), for all \( n \), \( \mathcal{E}_{M_d}(n; S) \geq c_\delta n^{\delta/(d-1)} \).

Let us decompose \( M_d \) into **slices** each of which is a copy of \( M_{d-1} \), by fixing the value in a single coordinate-position. By induction, any resulting slice that contains \( m \) **ELIGIBLE** nodes of \( M_d \) contains \( \mathcal{E}_{M_{d-1}}(m; S) \geq c_{d-1} m^{(d-1)/(d-2)} \) **EXECUTED** nodes. By focusing only on the mutually disjoint slices corresponding to fixing \( u_1 \) at the successive values 0, 1, 2, \ldots, in the light of the just-described inequalities, we infer that, for any schedule \( S \),

\[
\mathcal{E}_{M_d}(n; S) \geq c_{d-1} \cdot \min_{m_0, \ldots, m_r = n} \sum_{j=1}^{r} m_j^{d-1}. 
\]

(3.1)

The intention here is that, of the \( n \) **ELIGIBLE** nodes of \( M_d \), \( m_j \) appear in the slice \( u_1 = j \), for \( j = 0, 1, \ldots, r \). The requirement that parents of **ELIGIBLE** nodes be **EXECUTED** nodes, coupled with Fact 1, guarantees that \( m_0 > \cdots > m_r \), which in turn guarantees that \( m_0 > \tau \). By the concavity of the function \( x^{(d-1)/(d-2)} \), the sum in (3.1) is minimized when the \( m_j \) are as close to equal as possible. We are, thus, led to the following assignment of the desired \( n \) **ELIGIBLE** nodes to slices. We start with \( m_0 = \lfloor n^{1/(d-1)} \rfloor \) and each \( m_j = m_0 - j \). If the thus-assigned \( m_j \) sum to less than \( n \), then we increment \( m_0 \leftarrow m_0 + 1 \), \( m_1 \leftarrow m_1 + 1 \), \ldots, in turn, proceeding until we have a total of \( n \) **ELIGIBLE** nodes.

We thus have

\[
\mathcal{E}_{M_d}(n; S) \geq c_{d-1} \cdot \sum_{j=1}^{\lfloor n^{1/(d-1)} \rfloor} j^{d-1} = \Omega(n^{d/(d-1)}).
\]

(b)(ii) Any schedule \( S \) that allocates **ELIGIBLE** nodes of \( M_d \) along successive diagonal levels achieves the advertised upper bound. This is because the \( \binom{m+d-1}{d-1} \) nodes on level \( L_m^{(d)} \) of \( M_d \) are all **ELIGIBLE** whenever all of their

\[
\sum_{j=0}^{m-1} \binom{j+d-1}{d-1} = \binom{m+d-1}{d}
\]

ancestors have been **EXECUTED**. As schedule \( S \) proceeds from the position wherein the **ELIGIBLE** nodes comprise level \( L_m^{(d)} \), to the position wherein the **ELIGIBLE** nodes comprise level \( L_{m+1}^{(d)} \), the number of **EXECUTED** nodes always has the form \( \binom{m+d-1}{d} + \ell \) for some \( \ell \leq |L_{m+1}^{(d)}| - |L_m^{(d)}| \).
Figure 3. The start of a diagonal-level computation of $M_2$. (Note the snaked order.) “X” denotes an executed node; “E” denotes an eligible node.

\[
\binom{m+d-1}{d-1} \leq n < \binom{m+d}{d-1},
\]

for some $m \geq 0$. For any fixed $d$, therefore, we have $m = \Theta(n^{1/(d-1)})$, so that $E_{M_d}(n; S) = \Theta(n^{d/(d-1)})$.

4. Schedules for Mesh-Like Monoid Dags

We now extend the results of Section 3 to relatives of mesh-dags whose structures arise from certain families of abelian monoids. From an algebraic perspective, these dags are monoid-induced analogues of the familiar group-induced Cayley graphs. From a graph-theoretic perspective, these dags are somewhat distorted copies of $M_2$, enhanced with regularly placed “shortcut” arcs.

4.1. Monoid-induced dags

A large, decades-old literature amply illustrates the benefits, in many computational situations, of exploiting the algebraic structure of graphs, as manifested in Cayley digraphs of algebras [1, 6, 16]. A finitely generated monoid is a system $M = M(\Gamma, \cdot, 1)$, where:

- $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ is a finite set of generators;
- $\cdot$ is a binary associative multiplication;
- $1$ is a (two-sided) multiplicative identity: for all $\xi \in M$, we have $\xi \cdot 1 = 1 \cdot \xi = \xi$.

The elements of $M$ comprise all finite products of instances of $\Gamma \cup \{1\}$. As is common, we use “$M$” ambiguously, to denote both the monoid $M(\Gamma, \cdot, 1)$ and the set of its elements; context will always clarify our intention. The nodes of the Cayley digraph $G(M)$ associated with $M = M(\Gamma, \cdot, 1)$ are all elements of $M$; each arc has the form $(\xi \rightarrow \xi : \gamma_i)$ for some $\xi \in M$ and some $\gamma_i \in \Gamma$. Most of the computational literature on Cayley digraphs focuses on Cayley digraphs...
of groups (i.e., monoids with multiplicative inverses); however, major insights emerge from studying Cayley digraphs of monoids, cf. the studies of data structures in [16, 17] which inspire the current technical development.

The acyclicity required in computation-dags manifests itself as a natural restriction on the monoids underlying the Cayley digraphs of interest. The reader can easily verify the following.

Lemma 1 The Cayley digraph \( G(M) \) is acyclic if, and only if, \( M \) does not contain elements \( \xi \) and \( \eta \neq 1 \), such that \( \xi = \xi \cdot \eta \cdot \xi \). This can occur only when \( M \) is infinite.

Since we seek generalizations of mesh-dags, we henceforth focus on Cayley digraphs of abelian monoids, in which the following commutative identities hold.

\[
\gamma_1 \cdot \gamma_2 = \gamma_2 \cdot \gamma_1 \quad \text{for all generators } \gamma_1, \gamma_2. \tag{4.1}
\]

A monoid \( M \) with generator-set \( \Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\} \) is \( k \)-dimensional (where \( k \leq n \)) if there exists an \( k \)-element subset \( \Gamma' \subseteq \Gamma \) such that each \( \xi \in M - \{1\} \) is a product of elements of \( \Gamma' \). (The elements of \( \Gamma - \Gamma' \) supply "shortcuts" in \( G(M) \).) We call \( \Gamma' \) a \( k \)-basis for \( M \).

Mesh-Like Monoid Dags. We now delimit the computation-dags of interest. A Mesh-Like Monoid dag (MLM-dag, for short) \( M \) is the Cayley digraph of a finitely generated infinite abelian monoid, such that:

- \( M \) is acyclic (cf. Lemma 1);
- \( M \) has infinite width (in the sense of Section 2.3).

An MLM-dag is 2-dimensional (is a 2MLM-dag) if its underlying monoid is 2-dimensional.

Easily, every mesh-dag is (isomorphic to) an MLM-dag. Indeed, mesh-dags are the most restrictive MLM-dags, in that the identities (4.1) are the only nontrivial relations among their elements. Specifically, \( M_d \) is the MLM-dag based on the monoid \( M \) that has:

- \( d \) generators \( g_1, \ldots, g_d \in \mathbb{N}^d \); each \( g_i = \langle 0, \ldots, 0, 1, 0, \ldots, 0 \rangle \), with the sole 1 in position \( i \).
- coordinatewise addition as its "multiplication";
- the identity \( 1 = \langle 0, 0, \ldots, 0 \rangle \).

4.2. Delimiting the structure of 2MLM-dags

We now expose the facet of the structure of 2MLM-dags that allows us to bound the growth of their enabling functions. We focus on an arbitrary 2MLM-dag \( M \) that is based on an abelian monoid \( M \) having a 2-basis \( \{\lambda, \rho\} \).

Theorem 2 If \( M \) is a 2MLM-dag, then the basis generators \( \lambda, \rho \) of the monoid \( M \) underlying \( M \) do not obey any equation of the form\(^2\)

\[
\lambda^a \cdot \rho^b = \lambda^c \cdot \rho^d \tag{4.2}
\]

\(^2\)As usual, for any generator \( \gamma \), we have \( \gamma^0 = 1 \) and \( \gamma^{i+1} = \gamma \cdot \gamma^i \).

for nonnegative integers \( a, b, c, d \), except for the trivial equation wherein \( a = c \) and \( b = d \).

Proof Sketch. Assume that (4.2) holds for \( a, b, c, d \in \mathbb{N} \), where either \( a \neq c \) or \( b \neq d \) (or both). We expose the absurdity of this assumption via the following simple fact.

Fact 2 If \( \xi = \eta \) for \( \xi, \eta \in M \), then for all \( \zeta \in M \), we have \( \xi \cdot \zeta = \eta \cdot \zeta \).

We isolate three cases that exhaust the ways in which \( a \neq c \) or \( b \neq d \) (or both).

Case 1: \([a < c \text{ and } b \leq d]\) or \([a \leq c \text{ and } b < d]\).

In this case, we have \( \lambda^a \cdot \rho^b = (\lambda^a \cdot \rho^b \cdot (\lambda^{c-a} \cdot \rho^{d-b})) \), which contradicts \( M \)'s alleged acyclicity—since \((\lambda^{c-a} \cdot \rho^{d-b}) \in M - \{1\}\).

Case 2: \( a + b = c + d \).

In this case, nodes \( \lambda^a \cdot \rho^b \) and \( \lambda^c \cdot \rho^d \) lie on the same “diagonal” level \( \ell = a + b = c + d \) of \( M \). We can, therefore, rewrite (4.2) in the form \( \lambda^a \cdot \rho^{b-a} = \lambda^c \cdot \rho^{d-c} \).

Say, with no loss of generality, that \( a < c \), and let \( e = c - a \). Consider the population of level \( \ell \) of \( M \), as determined by the analogous set for \( M \). We find that:

- \( \leq a \) nodes at level \( \ell \) have the form \( \lambda^x \cdot \rho^\ell - x \), where \( 0 \leq x < a \).
- \( \leq \ell - c \) nodes have the form \( \lambda^x \cdot \rho^{\ell - x} \), where \( c < x \leq \ell \).
- There is node \( \lambda^a \cdot \rho^{\ell - a} = \lambda^c \cdot \rho^{\ell - c} \).
- \( \leq e - 1 \) nodes have the form \( \lambda^x \cdot \rho^{\ell - x} \), where \( a < x < e \).

Level \( \ell \) thus contains no more than \( e \) nodes. Consider next the population of level \( \ell + e \) of \( M \), as determined by the analogous set for \( M \). We find that:

- \( \leq a \) nodes at level \( \ell + e \) have the form \( \lambda^x \cdot \rho^{\ell + e - x} \), where \( 0 \leq x < a \).
- \( \leq \ell - c \) nodes have the form \( \lambda^x \cdot \rho^{\ell - x} \), where \( c < x \leq \ell \).
- There is node \( \lambda^a \cdot \rho^{\ell - a} = \lambda^c \cdot \rho^{\ell - c} \).

These equations follow from Fact 2 or direct calculation.

- We have the following two sets of nodes:
  - \( * \leq e - 1 \) nodes have the form \( \lambda^x \cdot \rho^{\ell - x} \), where \( a < x < a + e \).
  - \( * \leq e - 1 \) nodes have the form \( \lambda^x \cdot \rho^{\ell - x} \), where \( c < x < e + e \).

However, these two sets are identical!

Fact 3 On level \( \ell + e \) of \( M \), the set of nodes that lie "between" node \( \lambda^a \cdot \rho^{\ell + e - a} \) and node \( \lambda^a \cdot \rho^{\ell - a} \) is identical to the set of nodes that lie "between" node \( \lambda^c \cdot \rho^{\ell + e - c} \) and node \( \lambda^c \cdot \rho^{\ell - c} \).
Fact 3 is verified via the following system of equations. For each $j \in \{1, 2, \ldots, e-1\}$:

$$\lambda^{a+j} \cdot \rho^{e+r-(a+j)} = (\lambda^a \cdot \rho^{e-a}) \cdot (\lambda^j \cdot \rho^{r-j}) = (\lambda^e \cdot \rho^{e-c}) \cdot (\lambda^j \cdot \rho^{r-j}) = \lambda^{a+j} \cdot \rho^{e+r-(a+j)}$$ (4.3)

In toto, then, level $\ell + e$ contains no more than $\ell$ nodes. Fig. 4 illustrates the argument schematically.

One extends the reasoning leading to system (4.3) to show that each level $\ell + ke$ of $M$ contains no more than $\ell$ nodes. We conclude that $M$ has finite width; to wit:

**Fact 4** Say that there exist fixed constants $s$ and $k(s)$ such that, for all but finitely many lines of slope $s$ in the $\lambda\rho$ plane, a line segment of length $k(s)$ contains all of the nodes of the dag $M$ along that line. Then $M$ has finite width.

**Verification.** As in Theorem 1(a)(i), we note that two nodes of $M$, being a 2MLM-dag, have infinite width; hence, (4.2) cannot hold in this case.

Case 3: $a + b \neq c + d$.

This “skewed” version of Case 2 also implies that $M$ has finite width.

Cases 1–3 exhaust the nontrivial instances of equation (4.2), hence prove the theorem. ■

### 4.3. Nearly optimal schedules for 2MLM-dags

**Theorem 3** Let $\mathcal{M}$ be a 2MLM-dag with underlying monoid $M$ and 2-basis $\{\lambda, \rho\}$. (a) For any schedule $S$ for $\mathcal{M}$, $E_M(n; S) \geq \binom{n}{2}$. (b) Any schedule $S$ for $\mathcal{M}_2$ that schedules nodes along diagonal levels of the $\lambda\rho$ plane has $E_{\mathcal{M}_2}(n; S) \leq \binom{n}{2} + n$.

**Proof Sketch.** (a) Since $N_M = \{\lambda^a \cdot \rho^b \mid a, b \in \mathbb{N}\}$, we can invoke the proof of Theorem 1(a)(i) verbatim, with only the notational changes necessitated by the shift from mesh-dags to monoid-dags.  

(b) Let $\gamma$ be any generator of $M$ (hence, an arc-label of $\mathcal{M}$). By hypothesis, we must have $\gamma = \lambda^a \cdot \rho^b$ where $a + b > 0$. Now, if node $\lambda^a \cdot \rho^b$ of $M$ is ELIGIBLE at some step $t$ of the W-O Pebble Game, then all of the nodes $\{\lambda^u \cdot \rho^v \mid [u < x] \text{ and } [v < y]\}$ must be EXECUTED by step $t$. Thus, the ELIGIBLE status of node $\lambda^x \cdot \rho^y$ is unaffected by the presence or absence of an arc $(\lambda^{a-x} \cdot \rho^{b-y} \rightarrow \lambda^x \cdot \rho^y)$. (This arc would betoken the equation $\lambda^{a-x} \cdot \rho^{b-y} = \lambda^{x-a} \cdot \rho^{y-b} \cdot \gamma$ in $M$.) It follows that a node $\lambda^x \cdot \rho^y$ is ELIGIBLE at step $t$ of the W-O Pebble Game on the dag $M$ if, and only if, node $(x, y)$ is ELIGIBLE at step $t$ of the W-O Pebble Game on the mesh-dag $M_2$. Part (b) thus follows via the proof of Theorem 1(a)(ii), with only the notational changes necessitated by the shift to monoid-dags. ■

### 5. Conclusions

As the importance of the Internet as a platform for collaborative computing grows, the importance of understanding how to schedule computations effectively on this platform grows commensurately. We have taken a first step toward developing a theory that can yield this understanding. The W-O Pebble Game developed here models the process of scheduling a computation-dag on the Internet and provides a formal mechanism for comparing the quality of competing schedules. We have illustrated the use of our formalism by identifying a near-optimal class of schedules for computation-dags whose dependencies have the structure of a mesh of some finite dimensionality. By showing that these schedules retain their quality when used for a large variety of mesh-like computation-dags, we illustrate the applicability of our formal setting for a broader class of computations.

**Acknowledgment.** The author thanks Fran Berman for suggesting the problem studied here and Henri Casanova for helpful suggestions and comments.

### References


Figure 4. A schematic depiction of Fact 3. Node A is $\lambda^b \rho^d = \lambda^d \rho^b$; Node B is $\lambda^b \rho^{d+\epsilon} = (\lambda^d \rho^{d+\epsilon} = \lambda^d \rho^b) = \lambda^{b+\epsilon} \rho^d$.


[14] The Olson Laboratory Fight AIDS@Home project. [www.fightaidsathome.org].


