# Toward a Theory for Scheduling Dags in Internet-Based Computing

Grzegorz Malewicz, *Member*, *IEEE*,
Arnold L. Rosenberg, *Fellow*, *IEEE*, and Matthew Yurkewych

**Abstract**—Conceptual and algorithmic tools are developed as a foundation for a theory of scheduling complex computation-dags for Internet-based computing. The goal of the schedules produced is to render tasks eligible for allocation to remote clients (hence, for execution) at the maximum possible rate. This allows one to utilize remote clients well, as well as to lessen the likelihood of the "gridlock" that ensues when a computation stalls for lack of eligible tasks. Earlier work has introduced a formalism for studying this optimization problem and has identified optimal schedules for several significant families of structurally uniform dags. The current paper extends this work via a methodology for devising optimal schedules for a much broader class of complex dags, which are obtained via *composition* from a prespecified collection of simple building-block dags. The paper provides a suite of algorithms that decompose a given dag $\mathcal{G}$ to expose its building blocks and an execution-priority relation $\triangleright$ on building blocks. When the building blocks are appropriately interrelated under $\triangleright$, the algorithms specify an optimal schedule for $\mathcal{G}$.

**Index Terms**—Internet-based computing, grid computing, global computing, Web computing, scheduling dags, dag decomposition, theory.

## 1 Introduction

Earlier work [15], [17] has developed the Internet-Computing (IC, for short) Pebble Game, which abstracts the problem of scheduling computations having intertask dependencies,[1] for several modalities of Internet-based computing—including Grid computing (cf. [2], [6], [5]), global computing (cf. [3]), and Web computing (cf. [12]). The quality metric for schedules produced using the Game is to maximize the rate at which tasks are rendered eligible for allocation to remote clients (hence, for execution), with the dual aim of: 1) enhancing the effective utilization of remote clients and 2) lessening the likelihood of the "gridlock" that can arise when a computation stalls pending computation of already allocated tasks.

A simple example should illustrate our scheduling objective. Consider the two-dimensional evolving mesh of Fig. 1. An optimal schedule for this dag sequences tasks sequentially along each level [15] (as numbered in the figure). If just one client participates in the computation, then, after five tasks have been executed, we can allocate any of three eligible tasks to the client. If there are several clients, we could encounter a situation wherein two of these three eligible tasks (marked A in the figure) are allocated to clients who have not yet finished executing them. There is, then, only one task (marked E) that is eligible and unallocated. If two clients now request work, we may be able to satisfy only one request, thus wasting the computing resources of one client. Since an optimal schedule maximizes the number of eligible tasks, it minimizes the likelihood of this waste of resources (whose extreme case is the gridlock that arises when all eligible tasks have been allocated, but none has been executed).

Many IC projects—cf. [2], [11], [18]—monitor either the past histories of remote clients or their current computational capabilities or both. While the resulting snapshots yield no guarantees of future performance, they at least afford the server a basis for estimating such performance. Our study proceeds under the idealized assumption that such monitoring yields sufficiently accurate predictions of clients' future performance that the server can allocate eligible tasks to clients in an order that makes it likely that tasks will be executed in the order of their allocation. We show how such information often allows us to craft schedules that produce maximally many eligible tasks after each task execution.

**Our contributions**. We develop the framework of a theory of Internet-based scheduling via three conceptual/algorithmic contributions. 1) We introduce a new "priority" relation, denoted $\triangleright$, on pairs of bipartite dags; the assertion "$\mathcal{G}_1 \triangleright \mathcal{G}_2$" guarantees that one never sacrifices our quality metric (which rewards a schedule's rate of producing eligible tasks) by executing all sources of $\mathcal{G}_1$, then all sources of $\mathcal{G}_2$, then all sinks of both dags. We provide a repertoire of bipartite *building-block dags*, show how to schedule each optimally, and expose the $\triangleright$-interrelationships among them. 2) We specify a way of "composing" building blocks to obtain dags of possibly quite complex structures; cf. Fig. 2. If the building blocks used in the composition form a "relation-chain" under $\triangleright$, then the resulting composite dag

1. As is traditional—cf. [8], [9]—we model such a computation as a *dag* (*directed acyclic graph*).

• *G. Malewicz is with the Department of Engineering, Google, Inc., Mountain View, CA 94043. E-mail: malewicz@google.com.*
• *A.L. Rosenberg and M. Yurkewych are with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003. E-mail: {rsnbrg, yurk}@cs.umass.edu.*

*Manuscript received 14 Jan. 2005; revised 20 May 2005; accepted 20 June 2005; published online 21 Apr. 2006.*
*For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0008-0105.*

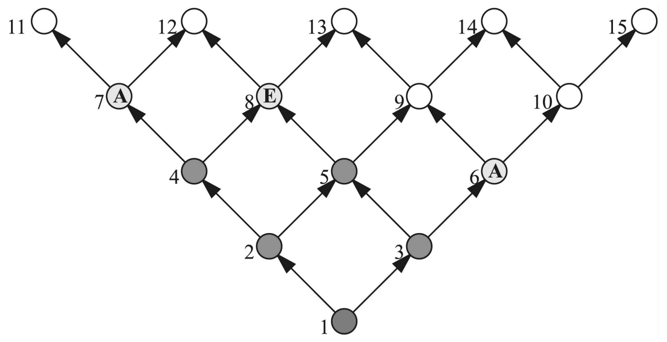

Fig. 1. An optimal schedule helps utilize clients well and reduce chances of gridlock.

is guaranteed to admit an optimal schedule. 3) The framework developed thus far is descriptive rather than prescriptive. It says that *if* a dag $\mathcal{G}$ is constructed from bipartite building blocks via composition and *if* we can identify the "blueprint" used to construct $\mathcal{G}$ and *if* the underlying building blocks are interrelated in a certain way, *then* a prescribed strategy produces an optimal schedule for $\mathcal{G}$. We next address the algorithmic challenge in the preceding *if*s: Given a dag $\mathcal{G}$, how does one apply the preceding framework to it? We develop a suite of algorithms that: a) reduce any dag $\mathcal{G}$ to its "transitive skeleton" $\mathcal{G}'$, a simplified version of $\mathcal{G}$ that shares the same set of optimal schedules; b) decompose $\mathcal{G}'$ to determine whether or not it is constructed from bipartite building blocks via composition, thereby exposing a "blueprint" for $\mathcal{G}'$; c) specify an optimal schedule for any such $\mathcal{G}'$ that is built from building blocks that form a "relation-chain" under $\triangleright$. For illustration, all of the dags in Fig. 2 yield to our algorithms.

The scheduling theory we develop here has the potential of improving efficiency and fault tolerance in existing Grid systems. As but one example, when Condor [19] executes computations with complex task dependencies, such as the Sloan Digital Sky Survey [1], it uses a "FIFO" regimen to sequence the allocation of eligible tasks. Given the temporal unpredictability of the remote clients, this scheduling may sometimes lead to an ineffective use of the clients' computing resources and, in the extreme case, to "gridlock." Our scheduling algorithms have the potential of reducing the severity of these issues. Experimental work is underway to determine how to enhance this potential.

**Related work**. Most closely related to our study are its immediate precursors and motivators, [15], [17]. The main results of those sources demonstrate the necessity and sufficiency of *parent orientation* for optimality in scheduling the dags of Fig. 3. Notably, these dags yield to the algorithms presented here, so our results both extend the results in [15], [17] and explain their underlying principles in a general setting. In a companion to this study, we are pursuing an orthogonal direction for extending [15], [17]. Motivated by the demonstration in Section 3.4 of the limited scope of the notion of optimal schedule that we study here, we formulate, in [14], a scheduling paradigm in which a server allocates *batches of tasks* periodically, rather than allocating individual tasks as soon as they become eligible. Optimality is always possible within this new framework,

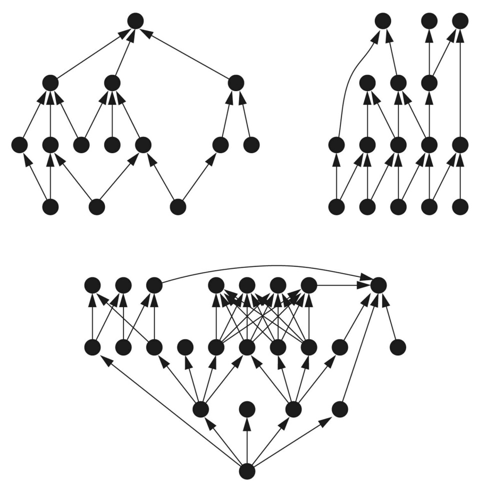
Fig. 2. Dags with complex task dependencies that our algorithms can schedule optimally.

but achieving it may entail a prohibitively complex computation. An alternative direction of inquiry appears in [7], [13], where a probabilistic pebble game is used to study the execution of interdependent tasks on unreliable clients. Finally, our study has been inspired by the many exciting systems and/or application-oriented studies of Internet-based computing, in sources such as [2], [3], [5], [6], [11], [12], [18].

## 2 Executing Dags on the Internet

We review the basic graph-theoretic terms used in our study. We then introduce several bipartite "building blocks" to exemplify our theory. Finally, we present the pebble game on dags we use to model computations on dags.

### 2.1 Computation-Dags

#### *2.1.1 Basic Definitions*

A *directed graph* $\mathcal{G}$ is given by a set of *nodes* $N_\mathcal{G}$ and a set of *arcs* (or, *directed edges*) $A_\mathcal{G}$, each having the form $(u \rightarrow v)$, where $u, v \in N_\mathcal{G}$. A *path* in $\mathcal{G}$ 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*) $\mathcal{G}$ is a directed graph that has no cycles, i.e., in a dag, no path of the preceding form has $u_1 = u_n$. When a dag $\mathcal{G}$ is used to model a computation, i.e., is a **computation-dag:**

- each $v \in N_\mathcal{G}$ represents a task in the computation;
- an arc $(u \rightarrow v) \in A_\mathcal{G}$ represents the dependence of task $v$ on task $u$: $v$ cannot be executed until $u$ is.

Given an arc $(u \rightarrow v) \in A_\mathcal{G}$, $u$ is a *parent* of $v$ and $v$ is a *child* of $u$ in $\mathcal{G}$. Each parentless node of $\mathcal{G}$ is a *source (node)*, and each childless node is a *sink (node)*; all other nodes are *internal*. A dag $\mathcal{G}$ is *bipartite* if:

1. $N_\mathcal{G}$ can be partitioned into subsets $X$ and $Y$ such that, for every arc $(u \rightarrow v) \in A_\mathcal{G}$, $u \in X$ and $v \in Y$;
2. each $v \in N_\mathcal{G}$ is *incident* to some arc of $\mathcal{G}$, i.e., is either the node $u$ or the node $w$ of some arc $(u \rightarrow w) \in A_\mathcal{G}$. (Prohibiting "isolated" nodes avoids degeneracies.)

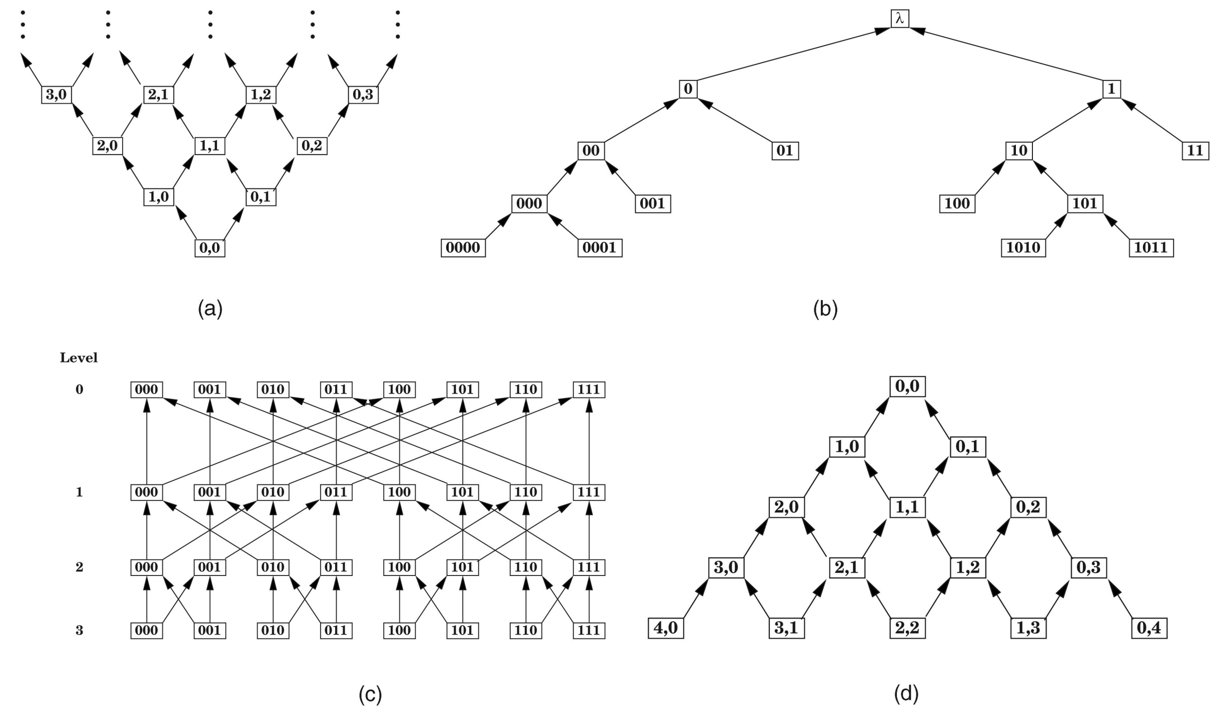

Fig. 3. (a) An evolving (two-dimensional) mesh, (b) a (binary) reduction-tree, (c) an FFT-dag, and (d) a (two-dimensional) reduction-mesh (or, pyramid dag).

$\mathcal{G}$ is *connected* if, when arc-orientations are ignored, there is a path connecting every pair of distinct nodes.

### 2.1.2 A Repertoire of Building Blocks

Our study applies to any repertoire of *connected bipartite building-block dags* that one chooses to build complex dags from. For illustration, we focus on the following specific dags. The following descriptions proceed left to right along successive rows of Fig. 4; we use the drawings to refer to "left" and "right."

The first three dags are named for the Latin letters suggested by their topologies. W-dags epitomize "expansive" and M-dags epitomize "reductive" computations.

**W-dags**. For each integer $d > 1$, the $(1,d)$-*W-dag* $\mathcal{W}_{1,d}$ has one source and $d$ sinks; its $d$ arcs connect the source to each sink. Inductively, for positive integers $a, b$, the $(a+b,d)$-W-dag $\mathcal{W}_{a+b,d}$ is obtained from the $(a,d)$-W-dag $\mathcal{W}_{a,d}$ and the $(b,d)$-W-dag $\mathcal{W}_{b,d}$ by identifying (or merging) the rightmost sink of the former dag with the leftmost sink of the latter.

**M-dags**. For each integer $d > 1$, the $(1,d)$-*M-dag* $\mathcal{M}_{1,d}$ has $d$ sources and one sink; its $d$ arcs connect each source to the sink. Inductively, for positive integers $a, b$, the $(a+b,d)$-M-dag $\mathcal{M}_{a+b,d}$ is obtained from the $(a,d)$-M-dag $\mathcal{M}_{a,d}$ and the $(b,d)$-M-dag $\mathcal{M}_{b,d}$ by identifying (or merging) the rightmost source of the former dag with the leftmost source of the latter.

**N-dags**. For each integer $s > 0$, the *s-N-dag* $\mathcal{N}_s$ has $s$ sources and $s$ sinks; its $2s-1$ arcs connect each source $v$ to sink $v$ and to sink $v+1$ if the latter exists. $\mathcal{N}_s$ is obtained from $\mathcal{W}_{s-1,2}$ by adding a new source on the right whose sole arc goes to the rightmost sink. The leftmost source of $\mathcal{N}_s$ —the dag's *anchor*—has a child that has no other parents.

**(Bipartite) Cycle-dags**. For each integer $s > 1$, the *s-(Bipartite) Cycle-dag* $\mathcal{C}_s$ is obtained from $\mathcal{N}_s$ by adding a new arc from the rightmost source to the leftmost sink—so that each source $v$ has arcs to sinks $v$ and $v+1 \bmod s$.

**(Bipartite) Clique-dags**. For each integer $s > 1$, the *s-(Bipartite) Clique-dag* $\mathcal{Q}_s$ has $s$ sources and $s$ sinks and an arc from each source to each sink.

We choose the preceding building blocks because the dags of Fig. 3 can all be constructed using these blocks. Although details must await Section 4, it is intuitively clear from the figure that the evolving mesh is constructed from its source outward by "composing" (or, "concatenating") a $(1,2)$-W-dag with a $(2,2)$-W-dag, then a $(3,2)$-W-dag, and so on; the reduction-mesh is constructed from its sources upward using $(k,2)$-M-dags for successively decreasing values of $k$; the reduction-tree is constructed from its sources/leaves upward by "concatenating" collections of $(1,2)$-M-dags; the FFT dag is constructed from its sources outward by "concatenating" collections of 2-cycles (which are identical to 2-cliques).

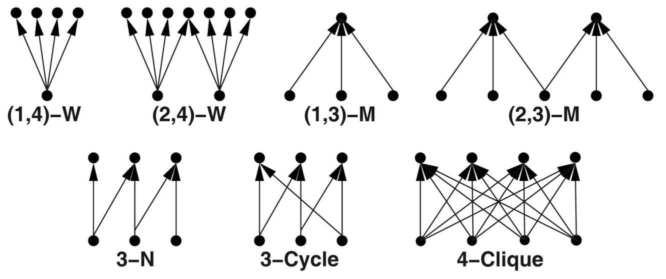

Fig. 4. The building blocks of semi-uniform dags.

## 2.2 The Internet-Computing Pebble Game

A number of so-called *pebble games* on dags have been shown, over the course of several decades, to yield elegant formal analogues of a variety of problems related to scheduling computation-dags. Such games use tokens, called *pebbles*, to model the progress of a computation on a dag: The placement or removal of the various available types of pebbles—which is constrained by the dependencies modeled by the dag's arcs—represents the changing (computational) status of the dag's task-nodes.

Our study is based on the Internet-Computing (IC, for short) Pebble Game of [15], whose structure derives from the "no recomputation allowed" pebble game of [16]. Arguments are presented in [15], [17] (q.v.) that justify studying a simplified form of the Game in which task-execution order follows task-allocation order. As we remark in the Introduction, while we recognize that this assumption will never be completely realized in practice, one hopes that careful monitoring of the clients' past behaviors and current capabilities, as prescribed in, say, [2], [11], [18], can enhance the likelihood, if not the certainty, of the desired order.

### 2.2.1 The Rules of the Game

The IC Pebble Game on a computation-dag $\mathcal{G}$ involves one player $S$, the *Server*, who has access to unlimited supplies of two types of pebbles: ELIGIBLE pebbles, whose presence indicates a task's eligibility for execution, and EXECUTED pebbles, whose presence indicates a task's having been executed. We now present the rules of our simplified version of the IC Pebble Game of [15], [17].

**The Rules of the IC Pebble Game**

- $S$ begins by placing an ELIGIBLE pebble on each unpebbled source of $\mathcal{G}$.
  /*Unexecuted sources are always eligible for execution, having no parents whose prior execution they depend on.*/
- At each step, $S$
  - selects a node that contains an ELIGIBLE pebble,
  - replaces that pebble by an EXECUTED pebble,
  - places an ELIGIBLE pebble on each unpebbled node of $\mathcal{G}$, all of whose parents contain EXECUTED pebbles.
- $S$'s goal is to allocate nodes in such a way that every node $v$ of $\mathcal{G}$ *eventually* contains an EXECUTED pebble.
  /*This modest goal is necessitated by the possibility that $\mathcal{G}$ is infinite.*/

**Note.** The (idealized) IC Pebble Game on a dag $\mathcal{G}$ executes one task/node of $\mathcal{G}$ per step. The reader should *not* infer that we are assuming a repertoire of tasks that are uniformly computable in unit time. Once we adopt the simplifying assumption that task-execution order follows task-allocation order, we can begin to measure time in an *event-driven* way, i.e., *per task,* rather than chronologically, i.e., per unit time. Therefore, our model allows tasks to be quite heterogeneous in complexity as long as the Server can match the tasks' complexities with the clients' resources (via the monitoring alluded to earlier).

A *schedule* for the IC Pebble Game on a dag $\mathcal{G}$ is a rule for selecting which ELIGIBLE pebble to execute at each step of a play of the Game. For brevity, we henceforth call a node ELIGIBLE (respectively, EXECUTED) when it contains an ELIGIBLE (respectively, an EXECUTED) pebble. For uniformity, we henceforth talk about executing nodes rather than tasks.

### 2.2.2 IC Quality

The goal in the IC Pebble Game is to play the Game in a way that maximizes the number of ELIGIBLE nodes at every step $t$. For each step $t$ of a play of the Game on a dag $\mathcal{G}$ under a schedule $\Sigma$: $\widehat{E}_\Sigma(t)$ denotes the number of nodes of $\mathcal{G}$ that are ELIGIBLE at step $t$ and $E_\Sigma(t)$ the number of ELIGIBLE *nonsource* nodes. (Note that $E_\Sigma(0) = 0$.)

*We measure the* **IC quality** *of a play of the IC Pebble Game on a dag $\mathcal{G}$ by the size of $\widehat{E}_\Sigma(t)$ at each step t of the play—the bigger $\widehat{E}_\Sigma(t)$ is, the better. Our goal is an* **IC-optimal** *schedule $\Sigma$, in which, for all steps t, $\widehat{E}_\Sigma(t)$ is as big as possible.*

It is not a priori clear that IC-optimal schedules ever exist! The property demands that there be a *single* schedule $\Sigma$ for dag $\mathcal{G}$ such that, at *every* step of the computation, $\Sigma$ maximizes the number of ELIGIBLE nodes across *all* schedules for $\mathcal{G}$. In principle, it could be that every schedule that maximizes the number of ELIGIBLE nodes at step $t$ requires that a certain set of $t$ nodes has been executed, while every analogous schedule for step $t + 1$ requires that a different set of $t + 1$ nodes has been executed. Indeed, we see in Section 3.4 that there exist dags that do not admit any IC-optimal schedule. Surprisingly, though, the strong requirement of IC optimality can be achieved for large families of dags—even ones of quite complex structure.

The significance of IC quality—hence of IC optimality—stems from the following intuitive scenarios: 1) Schedules that produce ELIGIBLE nodes maximally fast may reduce the chance of a computation's "stalling" because no new tasks can be allocated pending the return of already assigned ones. 2) If the Server receives a batch of requests for tasks at (roughly) the same time, then an IC-optimal schedule ensures that maximally many tasks are ELIGIBLE at that time so that maximally many requests can be satisfied. See [15], [17] for more elaborate discussions of IC quality.

## 3 The Rudiments of IC-Optimal Scheduling

We now lay the groundwork for an algorithmic theory of how to devise IC-optimal schedules. Beginning with a result that simplifies the quest for such schedules, we expose IC-optimal schedules for the building blocks of Section 2.1.2. We then create a framework for scheduling disjoint collections of building blocks via a priority relation on dags and we demonstrate the nonexistence of such schedules for certain other collections.

Executing a sink produces no ELIGIBLE nodes, while executing a nonsink may. This simple fact allows us to focus on schedules with the following simple structure:

**Lemma 1.** *Let $\Sigma$ be a schedule for a dag $\mathcal{G}$. If $\Sigma$ is altered to execute all of $\mathcal{G}$'s nonsinks before any of its sinks, then the IC quality of the resulting schedule is no less than $\Sigma$'s.*

When applied to a bipartite dag $\mathcal{G}$, Lemma 1 says that we never diminish IC quality by executing all of $\mathcal{G}$'s sources before executing any of its sinks.

### 3.1 IC-Optimal Schedules for Individual Building Blocks

A schedule for any of the very uniform dags of Fig. 3 is IC optimal when it sequences task execution sequentially along each level of the dags [15]. While such an order is neither necessary nor sufficient for IC optimality with the "semi-uniform" dags studied later, it is important when scheduling the building-block dags of Section 2.1.2.

**Theorem 1.** *Each of our building-block dags admits an IC-optimal schedule that executes sources from one end to the other; for N-dags, the execution must begin with the anchor.*

**Proof.** The structures of the building blocks render the following bounds on $E_\Sigma(t)$ obvious, as $t$ ranges from 0 to the number of sources in the given dag:[2]

$$\begin{array}{llll}
\mathcal{W}_{s,d} & : E_\Sigma(t) & \leq & (d-1)t + [t=s]; \\
\mathcal{N}_s & : E_\Sigma(t) & \leq & t; \\
\mathcal{M}_{s,d} & : E_\Sigma(t) & \leq & [t=0] + \lfloor (t-1)/(d-1) \rfloor; \\
\mathcal{C}_s & : E_\Sigma(t) & \leq & t - [t \neq 0] + [t=s]; \\
\mathcal{Q}_s & : E_\Sigma(t) & = & s \times [t=s].
\end{array}$$

The execution orders in the theorem convert each of these bounds to an equality. □

### 3.2 Execution Priorities for Bipartite Dags

We now define a relation on bipartite dags that often affords us an easy avenue toward IC-optimal schedules—for complex, as well as bipartite, dags.

Let the disjoint bipartite dags $\mathcal{G}_1$ and $\mathcal{G}_2$ have $s_1$ and $s_2$ sources and admit the IC-optimal schedules $\Sigma_1$ and $\Sigma_2$, respectively. If the following inequalities hold,[3]

$$\begin{aligned}
&(\forall x \in [0, s_1])\ (\forall y \in [0, s_2]) : \\
&E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \leq E_{\Sigma_1}(\min\{s_1, x+y\}) \\
&\qquad\qquad + E_{\Sigma_2}((x+y) - \min\{s_1, x+y\}),
\end{aligned} \tag{1}$$

then we say that $\mathcal{G}_1$ *has priority over* $\mathcal{G}_2$, denoted $\mathcal{G}_1 \rhd \mathcal{G}_2$.

The inequalities in (1) say that one never decreases IC quality by executing a source of $\mathcal{G}_1$, in preference to a source of $\mathcal{G}_2$, whenever possible.

The following result is quite important in our algorithmic framework:

**Theorem 2.** *The relation $\rhd$ on bipartite dags is transitive.*

**Proof.** Let $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$ be arbitrary bipartite dags such that:

1. each $\mathcal{G}_i$ has $s_i$ sources and admits an IC-optimal schedule $\Sigma_i$;
2. $\mathcal{G}_1 \rhd \mathcal{G}_2$ and $\mathcal{G}_2 \rhd \mathcal{G}_3$.

To see that $\mathcal{G}_1 \rhd \mathcal{G}_3$, focus on a moment when we have executed $x_1 < s_1$ sources of $\mathcal{G}_1$ and $x_3 \leq s_3$ sources of $\mathcal{G}_3$ (so $E_{\Sigma_1}(x_1) + E_{\Sigma_3}(x_3)$ sinks are ELIGIBLE). We consider two cases.

**Case 1.** $s_1 - x_1 \geq \min\{s_2, x_3\}$. In this case, we have

$$\begin{aligned}
E_{\Sigma_1}(x_1) + E_{\Sigma_3}(x_3) &\leq E_{\Sigma_1}(x_1) + E_{\Sigma_2}(\min\{s_2, x_3\}) \\
&\qquad + E_{\Sigma_3}(x_3 - \min\{s_2, x_3\}) \\
&\leq E_{\Sigma_1}(x_1 + \min\{s_2, x_3\}) \\
&\qquad + E_{\Sigma_3}(x_3 - \min\{s_2, x_3\});
\end{aligned} \tag{2}$$

the first inequality follows because $\mathcal{G}_2 \rhd \mathcal{G}_3$, the second because $\mathcal{G}_1 \rhd \mathcal{G}_2$. We can iterate these transfers until either all sources of $\mathcal{G}_1$ are EXECUTED or no sources of $\mathcal{G}_3$ are EXECUTED.

**Case 2.** $s_1 - x_1 < \min\{s_2, x_3\}$. This case is a bit subtler than the preceding one. Let $y = s_3 - x_3$ and $z = (s_1 - x_1) + (s_3 - x_3) = (s_1 - x_1) + y$. Then, $x_1 = s_1 - (z - y)$ and $x_3 = s_3 - y$. (This change of notation is useful because it relates $x_1$ and $x_3$ to the numbers of sources in $\mathcal{G}_1$ and $\mathcal{G}_3$.) We note the following useful facts about $y$ and $z$:

- $0 \leq y \leq z$ by definition,
- $0 \leq z < s_3$ because $s_1 - x_1 < x_3$,
- $z - y \leq s_1$ because $x_1 \geq 0$,
- $s_1 - x_1 = z - y$ by definition,
- $z - y < s_2$ because $s_1 - x_1 < s_2$.

Now, we apply these observations to the problem at hand. Because $\mathcal{G}_2 \rhd \mathcal{G}_3$ and $z - y \in [0, s_2]$ and $\{y, z\} \subseteq [0, s_3]$, we know that

$$E_{\Sigma_2}(s_2 - (z-y)) + E_{\Sigma_3}(s_3 - y) \leq E_{\Sigma_2}(s_2) + E_{\Sigma_3}(s_3 - z),$$

so that

$$E_{\Sigma_3}(s_3 - y) - E_{\Sigma_3}(s_3 - z) \leq E_{\Sigma_2}(s_2) - E_{\Sigma_2}(s_2 - (z-y)). \tag{3}$$

Intuitively, executing the last $z - y$ sources of $\mathcal{G}_2$ is no worse (in IC quality) than executing the "intermediate" sources $s_3 - z$ through $s_3 - y$ of $\mathcal{G}_3$.

Similarly, because $\mathcal{G}_1 \rhd \mathcal{G}_2$ and $z - y \in [0, \min\{s_1, s_2\}]$, we know that

$$E_{\Sigma_1}(s_1 - (z-y)) + E_{\Sigma_2}(s_2) \leq E_{\Sigma_1}(s_1) + E_{\Sigma_2}(s_2 - (z-y)),$$

so that

$$E_{\Sigma_2}(s_2) - E_{\Sigma_2}(s_2 - (z-y)) \ \leq E_{\Sigma_1}(s_1) - E_{\Sigma_1}(s_1 - (z-y)). \tag{4}$$

Intuitively, executing the last $z - y$ sources of $\mathcal{G}_1$ is no worse (in IC quality) than executing the last $z - y$ sources of $\mathcal{G}_2$.

By transitivity (of $\leq$ ), inequalities (3), (4) imply that

$$E_{\Sigma_3}(s_3 - y) - E_{\Sigma_3}(s_3 - z) \ \leq E_{\Sigma_1}(s_1) - E_{\Sigma_1}(s_1 - (z-y)),$$

so that

$$\begin{aligned}
E_{\Sigma_1}(x_1) + E_{\Sigma_3}(x_3) &= E_{\Sigma_1}(s_1 - (z-y)) + E_{\Sigma_3}(s_3 - y) \\
&\leq E_{\Sigma_1}(s_1) + E_{\Sigma_3}(s_3 - z) \\
&= E_{\Sigma_1}(s_1) + E_{\Sigma_3}(x_3 - (s_1 - x_1)).
\end{aligned} \tag{5}$$

The preceding cases—particularly, the chains of inequalities (2), (5)—verify that system (1) always holds for $\mathcal{G}_1$ and $\mathcal{G}_3$ so that $\mathcal{G}_1 \rhd \mathcal{G}_3$, as was claimed. □

Theorem 2 has a corollary that further exposes the nature of $\rhd$ and that tells us how to schedule pairwise $\rhd$-comparable

2. For any statement $P$ about $t$, $[P(t)] =$ **if** $P(t)$ **then** 1 **else** 0.
3. $[a, b]$ denotes the set of integers $\{a, a+1, \ldots, b\}$.

bipartite dags IC optimally. Specifically, we develop tools that extend Theorem 1 to disjoint unions—called *sums*—of building-block dags. Let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ be connected bipartite dags that are pairwise disjoint, in that $N_{\mathcal{G}_i} \cap N_{\mathcal{G}_j} = \emptyset$ for all distinct $i$ and $j$. The *sum* of these dags, denoted $\mathcal{G}_1 + \cdots + \mathcal{G}_n$, is the bipartite dag whose node-set and arc-set are, respectively, the unions of the corresponding sets of $\mathcal{G}_1, \ldots, \mathcal{G}_n$.

**Corollary 1.** *Let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ be pairwise disjoint bipartite dags, with each $\mathcal{G}_i$ admitting an IC-optimal schedule $\Sigma_i$. If $\mathcal{G}_1 \triangleright \cdots \triangleright \mathcal{G}_n$, then the schedule $\Sigma^\star$ for the sum $\mathcal{G}_1 + \cdots + \mathcal{G}_n$ that executes, in turn, all sources of $\mathcal{G}_1$ according to $\Sigma_1$, all sources of $\mathcal{G}_2$ according to $\Sigma_2$, and so on, for all $i \in [1, n]$, and, finally, executes all sinks, is IC optimal.*

**Proof.** By Lemma 1, we lose no generality by focusing on a step $t$ when the only EXECUTED nodes are sources of the sum-dag. For any indices $i$ and $j > i$, the transitivity of $\triangleright$ guarantees that $\mathcal{G}_i \triangleright \mathcal{G}_j$. Suppose that some sources of $\mathcal{G}_i$ are not EXECUTED at step $t$, but at least one source of $\mathcal{G}_j$ is EXECUTED. Then, by the definition of $\triangleright$, in (1), we never decrease the number of ELIGIBLE sinks at step $t$ by "transferring" as many source-executions as possible from $\mathcal{G}_j$ to $\mathcal{G}_i$. By repeating such "transfers" a finite number of times, we end up with a "left-loaded" situation at step $t$, wherein there exists $i \in [1, n]$ such that all sources of $\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}$ are EXECUTED, some sources of $\mathcal{G}_i$ are EXECUTED, and no sources of $\mathcal{G}_{i+1}, \ldots, \mathcal{G}_n$ are EXECUTED. □

One can actually prove Corollary 1 without invoking the transitivity of $\triangleright$ by successively "transferring executions" from each $\mathcal{G}_i$ to $\mathcal{G}_{i-1}$.

## 3.3 Priorities among Our Building Blocks

We now determine the pairwise priorities among the building-block dags of Section 2.1.2.

**Theorem 3.** *We observe the following pairwise priorities among our building-block dags:*

1. *For all $s$ and $d$, $\mathcal{W}_{s,d} \triangleright \mathcal{G}$ for the following bipartite dags $\mathcal{G}$:*
   a. *all W-dags $\mathcal{W}_{s',d'}$ whenever $d' < d$ or whenever $d' = d$ and $s' \geq s$;*
   b. *all M-dags, N-dags, and Cycle-dags; and*
   c. *Clique-dags $\mathcal{Q}_{s'}$ with $s' \leq d$.*
2. *For all $s$, $\mathcal{N}_s \triangleright \mathcal{G}$ for the following bipartite dags $\mathcal{G}$:*
   a. *all N-dags $\mathcal{N}_{s'}$, for all $s'$ and*
   b. *all M-dags.*
3. *For all $s$, $\mathcal{C}_s \triangleright \mathcal{G}$ for the following bipartite dags $\mathcal{G}$:*
   a. *$\mathcal{C}_s$ and*
   b. *all M-dags.*
4. *For all $s$ and $d$, $\mathcal{M}_{s,d} \triangleright \mathcal{M}_{s',d'}$ whenever $d' > d$ or whenever $d' = d$ and $s' \leq s$.*
5. *For all $s$, $\mathcal{Q}_s \triangleright \mathcal{Q}_s$.*

The proof of Theorem 3 is a long sequence of calculations paired with an invocation of the transitivity of $\triangleright$; we relegate it to the Appendix (Section A), which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm.

## 3.4 Incompatible Sums of Building Blocks

Each of our building blocks admits an IC-optimal schedule, but some of their sums do not.

**Lemma 2.** *The following sums of building-block dags admit no IC-optimal schedule:*

1. *all sums of the forms $\mathcal{C}_{s_1} + \mathcal{C}_{s_2}$ or $\mathcal{C}_{s_1} + \mathcal{Q}_{s_2}$ or $\mathcal{Q}_{s_1} + \mathcal{Q}_{s_2}$, where $s_1 \neq s_2$;*
2. *all sums of the form $\mathcal{N}_{s_1} + \mathcal{C}_{s_2}$ or $\mathcal{N}_{s_1} + \mathcal{Q}_{s_2}$; and*
3. *all sums of the form $\mathcal{Q}_{s_1} + \mathcal{M}_{s_2,d}$, where $s_1 > s_2$.*

**Proof.**

1. Focus on schedules for the dag $\mathcal{G} = \mathcal{C}_{s_1} + \mathcal{C}_{s_2}$, where $s_1 \neq s_2$. There is a unique family $\Phi_1$ of schedules for which $E_\Sigma(s_1) = s_1$; all of these execute sources of $\mathcal{C}_{s_1}$ for the first $s_1$ steps. For any other schedule $\Sigma'$, $E_{\Sigma'}(s_1) < E_\Sigma(s_1)$. Similarly, there is a unique family $\Phi_2$ of schedules for which $E_\Sigma(s_2) = s_2$; all of these execute sources of $\mathcal{C}_{s_2}$ for the first $s_2$ steps. For any other schedule $\Sigma'$, $E_{\Sigma'}(s_2) < E_\Sigma(s_2)$. Since $s_1 \neq s_2$, the families $\Phi_1$ and $\Phi_2$ are disjoint! Thus, no schedule for $\mathcal{G}$ maximizes IC quality at both steps $s_1$ and $s_2$; hence, $\mathcal{G}$ does not admit any IC-optimal schedule.

   Exactly the same argument works for the other indicated sum-dags of part 1.
2. Say, for contradiction, that there is an IC-optimal schedule $\Sigma$ for a dag $\mathcal{N}_{s_1} + \mathcal{G}_{s_2}$, where $\mathcal{G}_{s_2} \in \{\mathcal{C}_{s_2}, \mathcal{Q}_{s_2}\}$. The first node that $\Sigma$ executes must be the anchor of $\mathcal{N}_{s_1}$ for only this choice yields $E_\Sigma(1) \neq 0$. It follows that $\Sigma$ must execute all sources of $\mathcal{N}_{s_1}$ in the first $s_1$ steps, for this would yield $E_\Sigma(t) = t$ for all $t \leq s_1$, while any other choice would not maximize IC quality until step $s_1$. We claim that $\Sigma$ does not maximize IC quality at some step $s > 1$ and, hence, cannot be IC optimal. To wit: If $s_2 \leq s_1$, then $\Sigma$'s deficiency is manifest at step $s_1 + 1$. A schedule $\Sigma'$ that executes all sources of $\mathcal{G}_{s_2}$ and then executes $s_1 - s_2 + 1$ sources of $\mathcal{N}_{s_1}$ has $E_{\Sigma'}(s_1 + 1) = s_1 + 1$. But, $\Sigma$ executes a source of $\mathcal{G}_{s_2}$ for the first time at step $s_1 + 1$ and, so, $E_\Sigma(s_1 + 1) = s_1$. If $s_2 > s_1$, then $\Sigma$'s deficiency is manifest at step $s_2$. A schedule $\Sigma'$ that executes all sources of $\mathcal{G}_{s_2}$ during the first $s_2$ steps has $E_{\Sigma'}(s_2) = s_2$. However, during this period, $\Sigma$ executes some $x \geq 1$ sources of $\mathcal{N}_{s_1}$, hence, only some $y \leq s_2 - 1$ sources of $\mathcal{G}_{s_2}$. (Note that $x + y = s_2$.) Since $s_1 < s_2$, it must be that $y \geq 1$. But, then, by step $s_2$, $\Sigma$ will have produced exactly $x$ ELIGIBLE sinks on $\mathcal{N}_{s_1}$ and no more than $y - 1$ ELIGIBLE sinks on $\mathcal{G}_{s_2}$, so that $E_\Sigma(s_2) = x + y - 1 < s_2$.
3. Assume, for contradiction that there is an IC-optimal schedule $\Sigma$ for $\mathcal{Q}_{s_1} + \mathcal{M}_{s_2,d}$, where $s_1 > s_2$. Focus on the numbers of ELIGIBLE sinks after $s_1$ and after $s_2$ steps. The first $s_2$ nodes that $\Sigma$ executes must be nodes of $\mathcal{M}_{s_2,d}$ dictated by an IC-optimal

TABLE 1
The Relation $\rhd$ among Building-Block Dags

| $\mathcal{G}_1 \rhd \mathcal{G}_2$ | $\mathcal{W}_{s',d'}$ | $\mathcal{N}_{s'}$ | $\mathcal{M}_{s',d'}$ | $\mathcal{C}_{s'}$ | $\mathcal{Q}_{s'}$ |
|---|---|---|---|---|---|
| $\mathcal{W}_{s,d} \rhd$ | $d' < d$ **or** / $d' = d$ **and** $s' \geq s$ | all $s'$ | all $s', d'$ | all $s'$ | $s' \leq d$ **else** X |
| $\mathcal{N}_s \rhd$ | | all $s'$ | all $s', d'$ | X | X |
| $\mathcal{M}_{s,d} \rhd$ | | | $d' > d$ **or** / $d' = d$ **and** $s' \leq s$ | | X for $s' > s$ |
| $\mathcal{C}_s \rhd$ | | X | all $s', d'$ | $s' = s$ **else** X | X for $s' \neq s$ |
| $\mathcal{Q}_s \rhd$ | | X | X for $s > s'$ | X for $s' \neq s$ | $s' = s$ **else** X |

*Entries either list conditions for priority or indicate (via "X") the absence of any IC-optimal schedule for that pairing.*

schedule for that dag for this is the only choice for which $E_\Sigma(s_2) \neq 0$. A schedule $\Sigma'$ that executes all sources of $\mathcal{Q}_{s_1}$ during the first $s_1$ steps would have $E_{\Sigma'}(s_1) = s_1$. Consider what $\Sigma$ can have produced by step $s_1$. Since $\Sigma$ spends at least one step before step $s_1$ executing a node of $\mathcal{M}_{s_2,d}$, it cannot have rendered any sink of $\mathcal{Q}_{s_1}$ ELIGIBLE by step $s_1$; hence, $E_\Sigma(s_1) \leq \lfloor (s_1 - 1)/(d-1) \rfloor \leq s_1 - 1$. It follows that $\Sigma$ cannot be IC optimal. □

We summarize our priority-related results about sums of building blocks in Table 1.

## 4 On Scheduling Compositions of Building Blocks

We now show how to devise IC-optimal schedules for complex dags that are obtained via *composition* from any base set of connected bipartite dags that can be related by $\rhd$. We illustrate the process using the building blocks of Section 2.1.2 as a base set.

We inductively define the operation of *composition* on dags.

- Start with a base set **B** of connected bipartite dags.
- Given $\mathcal{G}_1, \mathcal{G}_2 \in$ **B**—which could be copies of the same dag with nodes renamed to achieve disjointness—one obtains a composite dag $\mathcal{G}$ as follows:
  - Let $\mathcal{G}$ begin as the sum, $\mathcal{G}_1 + \mathcal{G}_2$. Rename nodes to ensure that $N_\mathcal{G}$ is disjoint from $N_{\mathcal{G}_1}$ and $N_{\mathcal{G}_2}$.
  - Select some set $S_1$ of sinks from the copy of $\mathcal{G}_1$ in the sum $\mathcal{G}_1 + \mathcal{G}_2$ and an equal-size set $S_2$ of sources from the copy of $\mathcal{G}_2$.
  - Pairwise identify (i.e., merge) the nodes in $S_1$ and $S_2$ in some way.[4] The resulting set of nodes is $N_\mathcal{G}$; the induced set of arcs is $A_\mathcal{G}$.
- Add the dag $\mathcal{G}$ thus obtained to the set **B**.

We denote composition by $\Uparrow$ and say that the dag $\mathcal{G}$ is a *composite of type* $[\mathcal{G}_1 \Uparrow \mathcal{G}_2]$.

**Notes.** 1) The roles of $\mathcal{G}_1$ and $\mathcal{G}_2$ in a composition are asymmetric: $\mathcal{G}_1$ contributes sinks, while $\mathcal{G}_2$ contributes sources. 2) $\mathcal{G}'$s type indicates only that sources of $\mathcal{G}_2$ were merged with sinks of $\mathcal{G}_1$; it does not identify which nodes were merged. 3) The dags $\mathcal{G}_1$ and/or $\mathcal{G}_2$ could themselves be composite.

Composition is associative, so we do not have to keep track of the order in which dags are incorporated into a composite dag. Fig. 5 illustrates this fact, which we verify now.

**Lemma 3.** *The composition operation on dags is associative. That is, a dag $\mathcal{G}$ is a composite of type $[[\mathcal{G}_1 \Uparrow \mathcal{G}_2] \Uparrow \mathcal{G}_3]$ if, and only if, it is a composite of type $[\mathcal{G}_1 \Uparrow [\mathcal{G}_2 \Uparrow \mathcal{G}_3]]$.*

**Proof.** For simplicity, we refer to sinks and sources that are merged in a composition by their names prior to the merge. Context should disambiguate each occurrence of a name.

Let $\mathcal{G}$ be composite of type $[[\mathcal{G}_1 \Uparrow \mathcal{G}_2] \Uparrow \mathcal{G}_3]$, i.e., of type $[\mathcal{G}' \Uparrow \mathcal{G}_3]$, where $\mathcal{G}'$ is composite of type $[\mathcal{G}_1 \Uparrow \mathcal{G}_2]$. Let $T_1$ and $S_2$ comprise, respectively, the sinks of $\mathcal{G}_1$ and the sources of $\mathcal{G}_2$ that were merged to yield $\mathcal{G}'$. Note that no node from $T_1$ is a sink of $\mathcal{G}'$ because these nodes have become *internal* nodes of $\mathcal{G}'$. Let $T'$ and $S_3$ comprise, respectively, the sinks of $\mathcal{G}'$ and the sources of $\mathcal{G}_3$ that were merged to yield $\mathcal{G}$. Each sink of $\mathcal{G}'$ corresponds either to a sink of $\mathcal{G}_1$ that is not in $T_1$ or to a sink of $\mathcal{G}_2$. Hence, $T'$ can be partitioned into the sets $T'_1$, whose nodes are sinks of $\mathcal{G}_1$, and $T'_2$, whose nodes are sinks of $\mathcal{G}_2$. Let $S'_1$ and $S'_2$ comprise the sources of $\mathcal{G}_3$ that were merged with, respectively, nodes of $T'_1$ and nodes of $T'_2$. Now, $\mathcal{G}$ can be obtained by first merging the sources of $S'_2$ with the sinks of $T'_2$ and then merging the sources of the resulting dag, $S'_1 \cup S_2$, with the sinks, $T'_1 \cup T_1$, of $\mathcal{G}_1$. Thus,

4. When $S_1 = S_2 = \emptyset$, the composite dag is just a sum.

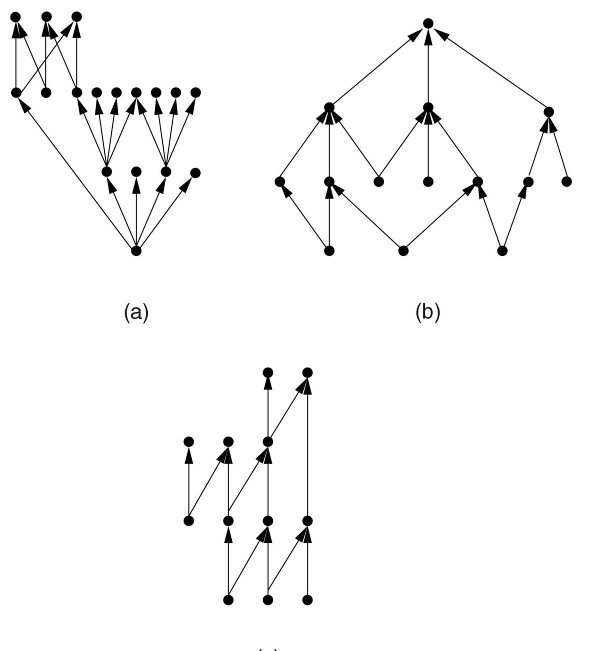

Fig. 5. Dags of the following types: (a) $[[\mathcal{W}_{1,5} \Uparrow \mathcal{W}_{2,4}] \Uparrow \mathcal{C}_3]$; (b) $[[[\mathcal{W}_{3,2} \Uparrow \mathcal{M}_{2,3}] \Uparrow \mathcal{M}_{1,2}] \Uparrow \mathcal{M}_{1,3}]$; (c) $[\mathcal{N}_3 \Uparrow [\mathcal{N}_3 \Uparrow \mathcal{N}_2]] = [[\mathcal{N}_3 \Uparrow \mathcal{N}_3] \Uparrow \mathcal{N}_2]$. Each admits an IC-optimal schedule.

$\mathcal{G}$ is also composite of type $[\mathcal{G}_1 \Uparrow [\mathcal{G}_2 \Uparrow \mathcal{G}_3]]$. The converse yields to similar reasoning. □

We can now illustrate the natural correspondence between the node-set of a composite dag and those of its building blocks, via Fig. 3:

- The evolving two-dimensional mesh is composite of type $\mathcal{W}_{1,2} \Uparrow \mathcal{W}_{2,2} \Uparrow \mathcal{W}_{3,2} \Uparrow \cdots$.
- A binary reduction-tree is obtained by pairwise composing of many instances of $\mathcal{M}_{1,2}$ (seven instances in the figure).
- The 5-level two-dimensional reduction-mesh is a composite of type $\mathcal{M}_{5,2} \Uparrow \mathcal{M}_{4,2} \Uparrow \mathcal{M}_{3,2} \Uparrow \mathcal{M}_{2,2} \Uparrow \mathcal{M}_{1,2}$.
- The FFT dag is obtained by pairwise composing many instances of $\mathcal{C}_2 = \mathcal{Q}_2$ (12 instances in the figure).

Dag $\mathcal{G}$ is a *$\triangleright$-linear composition* of the connected bipartite dags $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$ if:

1. $\mathcal{G}$ is a composite of type $\mathcal{G}_1 \Uparrow \mathcal{G}_2 \Uparrow \cdots \Uparrow \mathcal{G}_n$;
2. each $\mathcal{G}_i \triangleright \mathcal{G}_{i+1}$, for all $i \in [1, n-1]$.

Dags that are $\triangleright$-linear compositions admit simple IC-optimal schedules.

**Theorem 4.** *Let $\mathcal{G}$ be a $\triangleright$-linear composition of $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$, where each $\mathcal{G}_i$ admits an IC-optimal schedule $\Sigma_i$. The schedule $\Sigma$ for $\mathcal{G}$ that proceeds as follows is IC optimal:*

1. *$\Sigma$ executes the nodes of $\mathcal{G}$ that correspond to sources of $\mathcal{G}_1$, in the order mandated by $\Sigma_1$, then the nodes that correspond to sources of $\mathcal{G}_2$, in the order mandated by $\Sigma_2$, and so on, for all $i \in [1, n]$.*
2. *$\Sigma$ finally executes all sinks of $\mathcal{G}$ in any order.*

**Proof.** Let $\Sigma'$ be a schedule for $\mathcal{G}$ that has maximum $E_{\Sigma'}(x)$ for some $x$ and let $X$ comprise the first $x$ nodes that $\Sigma'$ executed. By Lemma 1, we may assume that either

1. $X$ contains all nonsinks of $\mathcal{G}$ (and perhaps some sinks) or
2. $X$ is a proper subset of the nonsinks of $\mathcal{G}$.

In situation 1, $E_\Sigma(x)$ is maximal by hypothesis. We therefore assume that situation 2 holds and show that $E_\Sigma(x) \geq E_{\Sigma'}(x)$. When $X$ contains only nonsinks of $\mathcal{G}$, each node of $X$ corresponds to a specific source of one specific $\mathcal{G}_i$. Let us focus, for each $i \in [1, n]$, on the set of sources of $\mathcal{G}_i$ that correspond to nodes in $X$; call this set $X_i$. We claim that:

*The number of ELIGIBLE nodes in $\mathcal{G}$ at step $x$, denoted $e(X)$, is $|S| - |X| + \sum_{i=1}^{m} e_i(X_i)$, where $S$ is the set of sources of $\mathcal{G}$, and $e_i(X_i)$ is the number of sinks of $\mathcal{G}_i$ that are ELIGIBLE when only sources $X_i$ of $\mathcal{G}_i$ are EXECUTED.*

To verify this claim, imagine that we execute nodes of $\mathcal{G}$ and the corresponding nodes of its building block $\mathcal{G}_i$ in tandem, using the terminology of the IC Pebble Game for convenience. The main complication arises when we pebble an *internal* node $v$ of $\mathcal{G}$ since we then simultaneously pebble a sink $v_i$ of some $\mathcal{G}_i$ and a source $v_j$ of some $\mathcal{G}_j$. At each step $t$ of the Game: If node $v$ of $\mathcal{G}$ becomes ELIGIBLE, then we place an ELIGIBLE pebble on $v_i$ and leave $v_j$ unpebbled; if $v$ becomes EXECUTED, then we place an EXECUTED pebble on $v_j$ and an ELIGIBLE pebble on $v_i$. An EXECUTED pebble on a sink of $\mathcal{G}$ is replaced with an ELIGIBLE pebble. No other pebbles change.

Focus on an arbitrary $\mathcal{G}_i$. Note that the sources of $\mathcal{G}_i$ that are EXECUTED comprise precisely the set $X_i$. The sinks of $\mathcal{G}_i$ that are ELIGIBLE comprise precisely the set $Y_i$ of sinks all of whose parents are EXECUTED; hence, $|Y_i| = e_i(X_i)$. The cumulative number of sources of the dags $\mathcal{G}_i$ that are ELIGIBLE is $|S| - p$, where $p$ is the number of sources of $\mathcal{G}$ that are EXECUTED. It follows that the cumulative number of ELIGIBLE pebbles on the dags $\mathcal{G}_i$ is $e_1(X_1) + \cdots + e_n(X_n) + |S| - p$. We now calculate the surfeit of ELIGIBLE pebbles on the dags $\mathcal{G}_i$ over the ELIGIBLE pebbles on $\mathcal{G}$. Extra ELIGIBLE pebbles get created when $\mathcal{G}$ is decomposed, in only two cases: 1) when an internal node of $\mathcal{G}$ becomes EXECUTED and 2) when we process a sink of $\mathcal{G}$ that is EXECUTED. The number of the former cases is $|X_1| + \cdots + |X_n| - p$. Denoting the number of the latter cases by $q$, we note that $q + |X_1| + \cdots + |X_n| = |X|$. The claim is thus verified because the number of ELIGIBLE nodes in $\mathcal{G}$ is

$$\begin{aligned} e(X) = {} & (e_1(X_1) + \cdots + e_n(X_n) + |S| - p) \\ & - (|X_1| + \cdots + |X_n| - p + q). \end{aligned}$$

Because of the priority relations among the dags $\mathcal{G}_i$, Corollary 1 implies that $e(X) = \sum_{i=1}^{n} E_{\Sigma_i}(x_i')$, where $x_i'$ is a "low-index-loaded" execution of the $\mathcal{G}_i$. Because of the way the dags $\mathcal{G}_i$ are composed, the sources of each $\mathcal{G}_j$ could have been merged only with sinks of lower-index dags, namely, $\mathcal{G}_1, \ldots, \mathcal{G}_{j-1}$. Thus, a "low-index-loaded" execution corresponds to a set $X'$ of $x$ EXECUTED nodes of $\mathcal{G}$ that satisfy precedence constraints. Thus, there is a schedule—namely, $\Sigma$—that executes nodes of $\mathcal{G}$ that correspond to the dags $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$, in turn, and this schedule is IC optimal. □

## 5 IC-Optimal Schedules via Dag-Decomposition

Section 4 describes how to build complex dags that admit IC-optimal schedules. Of course, the "real" problem is not to *build* a dag but rather to *execute* a given one. We now craft an algorithmic framework that converts the *synthetic* setting of Section 4 to an *analytical* setting. We present a suite of algorithms that take a given dag $\mathcal{G}$ and:

1. simplify $\mathcal{G}$'s structure in a way that preserves the IC quality of its schedules;
2. decompose (the simplified) $\mathcal{G}$ into its "constituents" (when it is, indeed, composite); and
3. determine when (the simplified) $\mathcal{G}$ is a $\triangleright$-linear composition of its "constituents."

When this program succeeds, we invoke Theorem 4 to schedule $\mathcal{G}$ IC optimally, bottom-up, from the decomposition. We now develop the advertised algorithmic setting.

### 5.1 "Skeletonizing" a Complex Dag

The word "simplified" is needed in the preceding paragraph because a dag can fail to be composite just because it contains "shortcut" arcs that do not impact intertask dependencies. Often, removing all shortcuts renders a dag composite, hence, susceptible to our scheduling strategy. (Easily, not every shortcut-free dag is composite.)

For any dag $\mathcal{G}$ and nodes $u, v \in N_{\mathcal{G}}$, we write $u \rightsquigarrow_{\mathcal{G}} v$ to indicate that there is a path from $u$ to $v$ in $\mathcal{G}$. An arc $(u \to v) \in A_{\mathcal{G}}$ is a *shortcut* if there is a path $u \rightsquigarrow_{\mathcal{G}} v$ that does not include the arc. The reader can show easily that:

**Lemma 4.** *Composite dags contain no shortcuts.*

Fortunately, one can efficiently remove all shortcuts from a dag without changing its set of IC-optimal schedules. A *(transitive) skeleton* (or, *minimum equivalent digraph*) $\mathcal{G}'$ of dag $\mathcal{G}$ is a smallest subdag of $\mathcal{G}$ that shares $\mathcal{G}$'s node-set and transitive closure [4].

**Lemma 5 ([10]).** *Every dag $\mathcal{G}$ has a unique transitive skeleton, $\sigma(\mathcal{G})$, which can be found in polynomial time.*

We can craft an IC-optimal schedule for a dag $\mathcal{G}$ automatically by crafting such a schedule for $\sigma(\mathcal{G})$. A special case of the following result appears in [15].

**Theorem 5.** *A schedule $\Sigma$ has the same IC quality when it executes a dag $\mathcal{G}$ as when it executes $\sigma(\mathcal{G})$. In particular, if $\Sigma$ is IC optimal for $\sigma(\mathcal{G})$, then it is IC optimal for $\mathcal{G}$.*

**Proof.** Say that, under schedule $\Sigma$, a node $u$ becomes ELIGIBLE at step $t$ of the IC Pebble Game on $\sigma(\mathcal{G})$. This means that, at step $t$, all of $u$'s ancestors in $\sigma(\mathcal{G})$—its parents, its parents' parents, etc.—are EXECUTED. Because $\sigma(\mathcal{G})$ and $\mathcal{G}$ have the same transitive closure, node $u$ has precisely the same ancestors in $\mathcal{G}$ as it does in $\sigma(\mathcal{G})$. Hence, under schedule $\Sigma$, $u$ becomes ELIGIBLE at step $t$ of the IC Pebble Game on $\mathcal{G}$. □

By Lemma 4, a dag cannot be composite unless it is transitively skeletonized. By Theorem 5, once having scheduled $\sigma(\mathcal{G})$ IC optimally, we have also scheduled $\mathcal{G}$ IC optimally. Therefore, this section paves the way for our decomposition-based scheduling strategy.

### 5.2 Decomposing a Composite Dag

Every dag $\mathcal{G}$ that is composed from connected bipartite dags can be decomposed to expose the dags and how they combine to yield $\mathcal{G}$. We describe this process in detail and illustrate it with the dags of Fig. 3.

A connected bipartite dag $\mathcal{H}$ is a **constituent** of $\mathcal{G}$ just when:

1. $\mathcal{H}$ is an *induced subdag* of $\mathcal{G}$: $N_{\mathcal{H}} \subseteq N_{\mathcal{G}}$ and $A_{\mathcal{H}}$ is comprised of all arcs $(u \to v) \in A_{\mathcal{G}}$ such that $\{u, v\} \subseteq N_{\mathcal{H}}$.
2. $\mathcal{H}$ is *maximal:* The induced subdag of $\mathcal{G}$ on any *superset* of $\mathcal{H}$'s nodes—i.e., any set $S$ such that $N_{\mathcal{H}} \subset S \subseteq N_{\mathcal{G}}$—is not connected and bipartite.

**Selecting a constituent.** We select any constituent of $\mathcal{G}$ all of whose sources are also sources of $\mathcal{G}$, if possible; we call the selected constituent $\mathcal{B}_1$ (the notation emphasizing that $\mathcal{B}_1$ is *bipartite*).

In Fig. 3: Every candidate $\mathcal{B}_1$ for the FFT dag is a copy of $\mathcal{C}_2$ included in levels 2 and 3; every candidate for the reduction-tree is a copy of $\mathcal{M}_{1,2}$; the unique candidate for the reduction-mesh is $\mathcal{M}_{4,2}$.

**Detaching a constituent.** We "detach" $\mathcal{B}_1$ from $\mathcal{G}$ by deleting the nodes of $\mathcal{G}$ that correspond to sources of $\mathcal{B}_1$, all incident arcs, and all resulting isolated sinks. We thereby replace $\mathcal{G}$ with a pair of dags $\langle \mathcal{B}_1, \mathcal{G}' \rangle$, where $\mathcal{G}'$ is the remnant of $\mathcal{G}$ after $\mathcal{B}_1$ is detached.

If $\mathcal{G}'$ is not empty, then the process of selection and detachment continues, producing a sequence of the form

$$\mathcal{G} \Longrightarrow \langle \mathcal{B}_1, \mathcal{G}' \rangle \Longrightarrow \langle \mathcal{B}_1, \langle \mathcal{B}_2, \mathcal{G}'' \rangle \rangle \Longrightarrow \langle \mathcal{B}_1, \langle \mathcal{B}_2, \langle \mathcal{B}_3, \mathcal{G}''' \rangle \rangle \rangle \Longrightarrow \cdots,$$

leading, ultimately, to a sequence of connected bipartite dags: $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$.

We claim that the described process recognizes whether or not $\mathcal{G}$ is composite and, if so, it produces the dags from which $\mathcal{G}$ is composed (possibly in a different order from the original composition). If $\mathcal{G}$ is not composite, then the process fails.

**Theorem 6.** *Let the dag $\mathcal{G}$ be composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$. The decomposition process produces a sequence $\mathcal{B}_1, \ldots, \mathcal{B}_n$ of connected bipartite dags such that:*

- *$\mathcal{G}$ is composite of type $\mathcal{B}_1 \Uparrow \cdots \Uparrow \mathcal{B}_n$;*
- *$\{\mathcal{B}_1, \ldots, \mathcal{B}_n\} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$.*

**Proof.** The result is trivial when $n = 1$ as $\mathcal{G}$ is then a connected bipartite dag. Assume, therefore, that the result holds for all $n < m$ and let $\mathcal{G}$ be a composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_m$. In this case, $\mathcal{G}_1$ is a constituent of $\mathcal{G}$, all of whose sources are sources of $\mathcal{G}$. (Other $\mathcal{G}_i$'s may share this property.) There is, therefore, a dag $\mathcal{B}_1$ for our process to detach. Since any constituent of $\mathcal{G}$ all of whose sources are sources of $\mathcal{G}$ must be one of the $\mathcal{G}_i$, we know that $\mathcal{B}_1$ is one of these dags. It follows that $\mathcal{G}$ is a composite of type

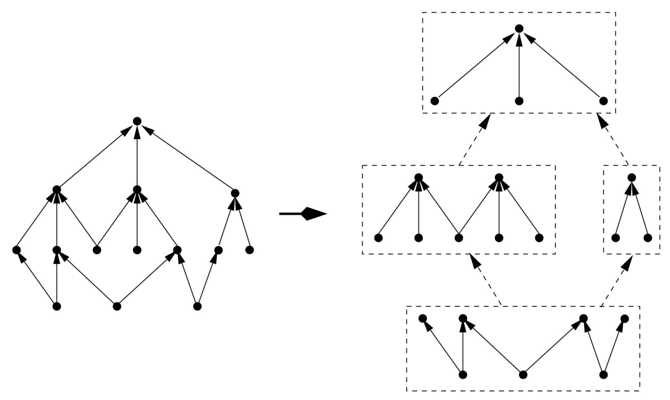

Fig. 6. The composition of the dags of Fig. 5b and its associated superdag.

$\mathcal{B}_1 \Uparrow (\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_{i-1} \Uparrow \mathcal{G}_{i+1} \Uparrow \cdots \Uparrow \mathcal{G}_m)$; moreover, the dag $\mathcal{G}'$ resulting after detaching $\mathcal{B}_1$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_{i-1} \Uparrow \mathcal{G}_{i+1} \Uparrow \cdots \Uparrow \mathcal{G}_m$ because the detachment process does not affect any sources of $\mathcal{G}$ other than those it shares with $\mathcal{B}_1$. By inductive hypothesis, then, $\mathcal{G}'$ can be decomposed as indicated in the theorem. We now invoke Lemma 3. □

### 5.3 The Super-Dag Obtained by Decomposing $\mathcal{G}$

The next step in our strategy is to abstract the structure of $\mathcal{G}$ exposed by its decomposition into $\mathcal{B}_1, \ldots, \mathcal{B}_n$ in an algorithmically advantageous way. Therefore, we shift focus from the decomposition to $\mathcal{G}$'s associated *super-dag* $\mathcal{S}_\mathcal{G} \overset{\text{def}}{=} \mathcal{S}(\mathcal{B}_1 \Uparrow \cdots \Uparrow \mathcal{B}_n)$, which is constructed as follows: Each node of $\mathcal{S}_\mathcal{G}$—which we call a *supernode* to prevent ambiguity—is one of the $\mathcal{B}_i$s. There is an arc in $\mathcal{S}_\mathcal{G}$ from supernode $u$ to supernode $v$ just when some sink(s) of $u$ are identified with some source(s) of $v$ when one composes the $\mathcal{B}_i$s to produce $\mathcal{G}$. Fig. 6 and Fig. 7 present two examples; in both, supernodes appear in dashed boxes and are interconnected by dashed arcs.

In terms of super-dags, the question of whether or not Theorem 4 applies to dag $\mathcal{G}$ reduces to the question of whether or not $\mathcal{S}_\mathcal{G}$ admits a *topological sort* [4] whose linearization of supernodes is consistent with the relation $\triangleright$. For instance, one derives an IC-optimal schedule for the dag $\mathcal{G}$ of Fig. 5b (which is decomposed in Fig. 6) by noting that $\mathcal{G}$ is a composite of type $\mathcal{W}_{3,2} \Uparrow \mathcal{M}_{1,2} \Uparrow \mathcal{M}_{2,3} \Uparrow \mathcal{M}_{1,3}$ and that $\mathcal{W}_{3,2} \triangleright \mathcal{M}_{1,2} \triangleright \mathcal{M}_{2,3} \triangleright \mathcal{M}_{1,3}$. Indeed, $\mathcal{G}$ points out the challenge in determining if Theorem 4 applies since it is also a composite of type $\mathcal{W}_{3,2} \Uparrow \mathcal{M}_{2,3} \Uparrow \mathcal{M}_{1,2} \Uparrow \mathcal{M}_{1,3}$, but $\mathcal{M}_{2,3} \not\triangleright \mathcal{M}_{1,2}$. We leave to the reader the easy verification that the linearization $\mathcal{B}_1, \ldots, \mathcal{B}_n$ is a topological sort of $\mathcal{S}(\mathcal{B}_1 \Uparrow \cdots \Uparrow \mathcal{B}_n)$.

### 5.4 On Exploiting Priorities among Constituents

Our remaining challenge is to devise a topological sort of $\mathcal{S}_\mathcal{G}$ that linearizes the supernodes in an order that honors relation $\triangleright$. We now present sufficient conditions for this to occur, verified via a linearization algorithm:

**Theorem 7.** *Say that the dag $\mathcal{G}$ is a composite of type $\mathcal{B}_1 \Uparrow \cdots \Uparrow \mathcal{B}_n$ and that, for each pair of constituents, $\mathcal{B}_i$, $\mathcal{B}_j$ with $i \neq j$, either $\mathcal{B}_i \triangleright \mathcal{B}_j$ or $\mathcal{B}_j \triangleright \mathcal{B}_i$. Then, $\mathcal{G}$ is a $\triangleright$-linear composition whenever the following holds:*

$$\begin{aligned}&\textit{Whenever } \mathcal{B}_j \textit{ is a child of } \mathcal{B}_i \textit{ in } \mathcal{S}(\mathcal{B}_1 \Uparrow \cdots \Uparrow \mathcal{B}_n),\\ &\textit{we have } \mathcal{B}_i \triangleright \mathcal{B}_j.\end{aligned} \quad (6)$$

**Proof.** We begin with an arbitrary topological sort, $\widehat{\mathcal{B}} \overset{\text{def}}{=} \mathcal{B}_{\alpha(1)}, \ldots, \mathcal{B}_{\alpha(n)}$, of the superdag $\mathcal{S}_\mathcal{G}$. We invoke the hypothesis that $\triangleright$ is a (weak) order on the $\mathcal{B}_i$'s to reorder $\widehat{\mathcal{B}}$ according to $\triangleright$, using a *stable*[5] comparison sort. Let $\vec{\mathcal{B}} \overset{\text{def}}{=} \mathcal{B}_{\beta(1)} \triangleright \cdots \triangleright \mathcal{B}_{\beta(n)}$ be the linearization of $\mathcal{S}_\mathcal{G}$ produced by the sort. We claim that $\vec{\mathcal{B}}$ is also a topological sort of $\mathcal{S}_\mathcal{G}$. To wit, pick any $\mathcal{B}_i$ and $\mathcal{B}_j$ such that $\mathcal{B}_j$ is $\mathcal{B}_i$'s child in $\mathcal{S}_\mathcal{G}$. By definition of topological sort, $\mathcal{B}_i$ precedes $\mathcal{B}_j$ in $\widehat{\mathcal{B}}$. We claim that, because $\mathcal{B}_i \triangleright \mathcal{B}_j$ (by (6)), $\mathcal{B}_i$ precedes $\mathcal{B}_j$ also in $\vec{\mathcal{B}}$. On the one hand, if $\mathcal{B}_j \not\triangleright \mathcal{B}_i$, then the sort necessarily places $\mathcal{B}_i$ before $\mathcal{B}_j$ in $\vec{\mathcal{B}}$. On the other hand, if $\mathcal{B}_j \triangleright \mathcal{B}_i$, then, since the sort is stable, $\mathcal{B}_i$ precedes $\mathcal{B}_j$ in $\vec{\mathcal{B}}$ because it precedes $\mathcal{B}_j$ in $\widehat{\mathcal{B}}$. Thus, $\vec{\mathcal{B}}$ is, indeed, a topological sort of $\mathcal{S}_\mathcal{G}$ so that $\mathcal{G}$ is composite of type $B_{\beta(1)} \Uparrow \cdots \Uparrow \mathcal{B}_{\beta(n)}$. In other words, $\mathcal{G}$ is the desired $\triangleright$-linear composition of $\mathcal{B}_{\beta(1)}, \ldots, \mathcal{B}_{\beta(n)}$. □

We can finally apply Theorem 4 to find an IC-optimal schedule for the dag $\mathcal{G}$.

## 6 Conclusions and Projections

We have developed three notions that form the basis for a theory of scheduling complex computation-dags for Internet-based computing: the *priority* relation $\triangleright$ on bipartite dags (Section 3.2), the operation of the *composition* of dags (Section 4), and the operation of the *decomposition* of dags (Section 5). We have established a way of combining these notions to produce schedules for a large class of complex computation-dags that maximize the number of tasks that are eligible for allocation to remote clients at every step of the schedule (Theorems 4 and 7). We have used our notions to progress beyond the structurally uniform computation-dags studied in [15], [17] to families that are built in structured, yet flexible, ways from a repertoire of bipartite building-block dags. The composite dags that we can now schedule optimally encompass not only those studied in [15], [17], but, as illustrated in Fig. 5, also dags that have rather complex structures, including nodes of varying degrees and nonleveled global structure.

One direction for future work is to extend the repertoire of building-block dags that form the raw material for our composite dags. In particular, we want building blocks of more complex structures than those of Section 2.1.2, including less-uniform bipartite dags and nonbipartite dags. We expect the computational complexity of our scheduling algorithms to increase with the structural complexity of our building blocks. Along these lines, we have thus far been unsuccessful in determining the complexity of the problem of deciding if a given computation-dag admits an IC-optimal schedule, but we continue to probe in this direction. (The scheduling problem could well be co-NP-Complete because of its underlying universal quantification.) Finally, we are working

5. That is, if $\mathcal{B}_i \triangleright \mathcal{B}_j$ and $\mathcal{B}_j \triangleright \mathcal{B}_i$, then the sort maintains the original relative order of $\mathcal{B}_i$ and $\mathcal{B}_j$.

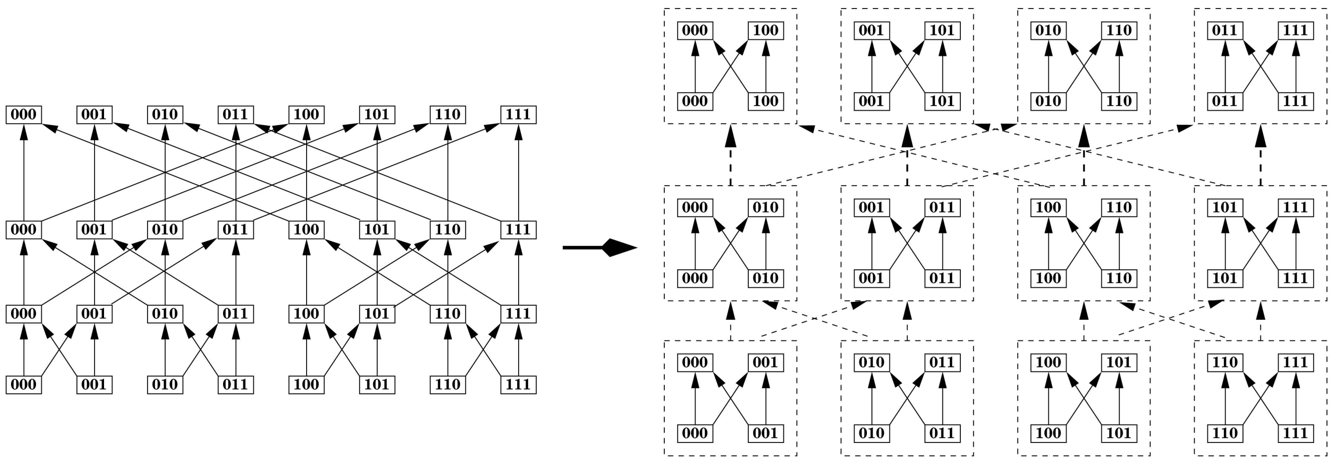
Fig. 7. The three-dimensional FFT dag and its associated superdag.

to extend Theorems 4 and 7 to loosen the strict requirement that the composite dag be a ▷-linear composition.

## ACKNOWLEDGMENTS

A portion of the research of G. Malewicz was done while he was visiting the TAPADS Group at the University of Massachusetts Amherst. The research of A. Rosenberg and M. Yurkewych was supported in part by US National Science Foundation Grant CCF-0342417. A portion of this paper appeared in the *Proceedings of the International Parallel and Distributed Processing Symposium*, 2005.

## REFERENCES

[1] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster, "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey," *Proc. 15th Conf. High Performance Networking and Computing,* p. 56, 2002.

[2] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service Oriented Grid Computing," *Proc. 10th Heterogeneous Computing Workshop,* 2001.

[3] W. Cirne and K. Marzullo, "The Computational Co-Op: Gathering Clusters into a Metacomputer," *Proc. 13th Int'l Parallel Processing Symp.,* pp. 160-166, 1999.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. Cambridge, Mass.: MIT Press, 2001.

[5] *The Grid: Blueprint for a New Computing Infrastructure,* second ed., I. Foster and C. Kesselman, eds. San Francisco: Morgan Kaufmann, 2004.

[6] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. High Performance Computing Applications,* vol. 15, pp. 200-222, 2001.

[7] L. Gao and G. Malewicz, "Internet Computing of Tasks with Dependencies Using Unreliable Workers," *Thoery of Computing Systems,* to appear.

[8] A. Gerasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling Dags on Multiprocessors," *J. Parallel and Distributed Computing,* vol. 16, pp. 276-291, 1992.

[9] L. He, Z. Han, H. Jin, L. Pan, and S. Li, "DAG-Based Parallel Real Time Task Scheduling Algorithm on a Cluster," *Proc. Int'l Conf. Parallel and Distruted Processing Techniques and Applications,* pp. 437-443, 2000.

[10] H.T. Hsu, "An Algorithm for Finding a Minimal Equivalent Graph of a Digraph," *J. ACM,* vol. 22, pp. 11-16, 1975.

[11] D. Kondo, H. Casanova, E. Wing, and F. Berman, "Models and Scheduling Guidelines for Global Computing Applications," *Proc. Int'l Parallel and Distruted Processing Symp.,* p. 79, 2002.

[12] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home: Massively Distributed Computing for SETI," *Computing in Science and Eng.,* P.F. Dubois, ed., Los Alamitos, Calif.: IEEE CS Press, 2000.

[13] G. Malewicz, "Parallel Scheduling of Complex Dags under Uncertainty," *Proc. 17th ACM Symp. Parallelism in Algorithms and Architectures,* 2005.

[14] G. Malewicz and A.L. Rosenberg, "On Batch-Scheduling Dags for Internet-Based Computing," *Proc. 11th European Conf. Parallel Processing,* 2005.

[15] A.L. Rosenberg, "On Scheduling Mesh-Structured Computations for Internet-Based Computing," *IEEE Trans. Computers,* vol. 53, pp. 1176-1186, 2004.

[16] A.L. Rosenberg and I.H. Sudborough, "Bandwidth and Pebbling," *Computing,* vol. 31, pp. 115-139, 1983.

[17] A.L. Rosenberg and M. Yurkewych, "Guidelines for Scheduling Some Common Computation-Dags for Internet-Based Computing," *IEEE Trans. Computers,* vol. 54, pp. 428-438, 2005.

[18] X.-H. Sun and M. Wu, "Grid Harvest Service: A System for Long-Term, Application-Level Task Scheduling," *Proc. IEEE Int'l Parallel and Distributed Processing Symp.,* p. 25, 2003.

[19] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience,* 2005.

**Grzegorz Malewicz** studied computer science and applied mathematics at the University of Warsaw from 1993 until 1998. He then joined the University of Connecticut and received the doctorate in 2003. He is a software engineer at Google, Inc. Prior to joining Google, he was an assistant professor of computer science at the University of Alabama (UA), where he taught computer science from 2003-2005. He has had internships at the AT&T Shannon Lab (summer 2001) and Microsoft Corporation (summer 2000 and fall 2001). He visited the Laboratory for Computer Science, MIT (AY 2002/2003) and was a visiting scientist at the University of Massachusetts Amherst (summer 2004) and Argonne National Laboratory (summer 2005). His research focuses on parallel and distributed computing, algorithms, combinatorial optimization and scheduling. His research appears in top journals and conferences and includes a *SIAM Journal of Computing* paper for which he was the sole author that solves a decade-old problem in distributed computing. He is a member of the IEEE.

**Arnold L. Rosenberg** is a Distinguished University Professor of Computer Science at the University of Massachusetts Amherst, where he codirects the Theoretical Aspects of Parallel and Distributed Systems (TAPADS) Laboratory. Prior to joining UMass, he was a professor of computer science at Duke University from 1981 to 1986 and a research staff member at the IBM T.J. Watson Research Center from 1965 to 1981. He has held visiting positions at Yale University and the University of Toronto; he was a Lady Davis Visiting Professor at the Technion (Israel Institute of Technology) in 1994, and a Fulbright Research Scholar at the University of Paris-South in 2000. His research focuses on developing algorithmic models and techniques to deal with the new modalities of "collaborative computing" (the endeavor of having several computers cooperate in the solution of a single computational problem) that result from emerging technologies. He is the author or coauthor of more than 150 technical papers on these and other topics in theoretical computer science and discrete mathematics and is the coauthor of the book *Graph Separators, with Applications*. He is a fellow of the ACM, a fellow of the IEEE, and a Golden Core member of the IEEE Computer Society.

**Matthew Yurkewych** received the BS degree from the Massachusetts Institute of Technology in 1998. He is a PhD Student in computer science at the University of Massachusetts-Amherst. Prior to entering graduate school, he worked at Akamai Technologies and CNet Networks as a software engineer.