On Scheduling Complex Dags for Internet-Based Computing
(Extended Abstract)

Grzegorz Malewicz  
Department of Computer Science  
University of Alabama  
Tuscaloosa, AL 35487, USA  
greg@cs.ua.edu

Arnold L. Rosenberg and Matthew Yurkewych  
Department of Computer Science  
University of Massachusetts Amherst  
Amherst, MA 01003, USA  
{rsnbrg,yurk}@cs.umass.edu

Abstract

Conceptual tools are developed to aid in crafting 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, and also 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. These dags are obtained via composition from a specified collection of simple building-block dags. The paper introduces a suite of algorithms that decompose a given dag to expose its building-blocks, and a priority relation on building-blocks. When the building-blocks are appropriately interrelated, the dag can be scheduled optimally.

1. Introduction

Earlier work [13, 15] has developed the Internet-Computing (IC, for short) Pebble Game, a variant of the classical pebble games that abstracts the problem of scheduling computations having intertask dependencies, for several modalities of Internet-based computing—including Grid computing (cf. [1, 6, 5]), global computing (cf. [2]), and Web computing (cf. [10]). 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: (a) enhancing the effective utilization of remote clients and (b) lessening the likelihood of the “gridlock” that can arise when a computation stalls pending computation of already-allocated tasks.

The main results of [13, 15] are schedules for the computation-dags of Fig. 1, that are optimal with respect to an idealized version of the IC Pebble Game. (The idealization resides in the assumption that tasks are executed in the order of their allocation. While this assumption will never be completely realized in practice, one hopes that careful monitoring of clients’ past behaviors and current capabilities, as prescribed in, say, [1, 9, 16] can enhance the likelihood, if not the certainty, of the desired order.) The current study is devoted to extrapolating on the scheduling techniques introduced in [13, 15], with the goal of developing a theory of scheduling complex computation-dags for Internet-based computing.

Our contributions. We assume that we have access to a repertoire of bipartite building-block dags that we know how to schedule optimally (according to a quality metric that rewards a schedule’s rate of producing eligible nodes). 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 “$G_1 \triangleright G_2$” guarantees that one never sacrifices quality by executing all sources of $G_1$, then all sources of $G_2$, then all sinks of both dags. We then investigate the $\triangleright$-interrelationships among our building-blocks. (2) We specify a way of “composing” building-blocks, to obtain dags of possibly quite complex structures. If the building blocks used in the composition are pairwise comparable under $\triangleright$, then the resulting composite dag is guaranteed to admit an optimal schedule. Notably, the computation-dags of Fig. 1 can be constructed via such composition, so our results both extend the results in [13, 15] and explain their underlying principles in a general setting. (3) The framework developed thus far is descriptive.

---

1 All technical terms are defined in Section 2.
cy of parent orientation for optimality in schedules for the
dags of Fig. 1. In a companion to this study, we are purs-
suing an orthogonal direction for extending [13, 15]. Moti-
vated by the demonstration in Section 3.1.C of the limited
scope of the notion of Internet-oriented schedule that we
study here, we formulate in [11] a scheduling paradigm in
which a server allocates batches of tasks periodically, rather
than allocating individual tasks as soon as they become eli-
gible. Optimality is always possible within this new frame-
work, but achieving it may entail a prohibitively complex
computation. An alternative direction of inquiry was stud-
ied in [7], where a probabilistic pebble game is used to study
the problem of executing tasks on unreliable clients.

Although the goals and methodology of our study dif-
fer significantly from those of [3, 12, 14], we owe an intel-
lectual debt to those pioneering studies of pebbling-based
scheduling models. Finally, our study has been inspired
by the many exciting systems- and/or application-oriented
studies of Internet-based computing, in sources such as
[1, 2, 5, 6, 9, 10, 16].

2. Executing Dags on the Internet

2.1. Computation-Dags

A. Basic definitions. A directed graph $G$ 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 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 $G$ is used to model a computa-
tion, i.e., is a computation-dag:

- each $v \in N_G$ represents a task in the computation;
- an arc $(u \rightarrow v) \in A_G$ represents the dependence of
task $v$ on task $u$; $v$ cannot be executed until $u$ is.

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

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

$G$ is connected if, when arc-orientations are ignored, there
is a path connecting every pair of distinct nodes. A con-
nected bipartite dag $H$ is a constituent of $G$ just when:

1. $H$ is an induced subdag of $G$: $N_H \subseteq N_G$, and $A_H$ com-
   prises all arcs $(u \rightarrow v) \in A_G$ such that $\{u, v\} \subseteq N_G$. rather than prescriptive. It says that if a computation-dag
$G$ is constructed from bipartite building blocks via composi-
tion, and if we can identify the “blueprint” used to con-
struct $G$, and if the underlying building blocks are interre-
lated in a certain way, then a prescribed strategy produces
an optimal schedule for $G$. We next address the algorithmic
challenge hidden in the preceding if’s: given a computation-
dag $G$, how does one apply the preceding framework to it?
We develop a suite of algorithms that: (a) reduce any
computation-dag $G$ to its “transitive skeleton” $G'$, a simpli-
fied version of $G$ that shares the same set of optimal sched-
ules; (b) decompose $G'$ to determine whether or not it is
constructed from bipartite building blocks via composition,
thereby exposing a “blueprint” for $G'$; (c) specify an op-
timal schedule for any such $G'$ that is built from building
blocks that are interrelated under $\triangleright$.

Related work. Most closely related to our study are its
immediate precursors and motivators, [13, 15]. The main
results of those sources demonstrate the necessity and suffi-
2. \( H \) is maximal: the induced subdag of \( G \) on any super-
set of \( H \)'s nodes—i.e., any set \( S \) such that \( N_H \subset S \subseteq N_G \)—is not connected and bipartite.

B. A repertoire of building-block dags. Our study applies
to any repertoire of connected bipartite building-block dags
that one chooses to build complex dags from. For illus-
tration, though, we focus on the following specific build-
ing blocks. The following descriptions proceed left to right
along successive rows of Fig. 2. For all descriptions, we use
the drawings in Fig. 2 to refer to “left” and “right.”

\[
(1,4) - W: \quad (2,4) - W: \quad (1,3) - M:\n\]
\[
(2,3) - M: \quad 3 - N: \quad 3 - Cycle:\n\]
\[
(3,4) - Clique:\quad (4,3) - Clique:\n\]

**Figure 2. The building blocks of semi-uniform dags.**

The first three dags are named for the Latin letters sug-
gested by their topologies.

**W-dags.** For each integer \( d > 1 \), the \((1, d)\)-W-dag
\( W_{1,d} \) 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 \( W_{a+b,d} \) is obtained from the \((a, d)\)-W-dag \( W_{a,d} \) and the \((b, d)\)-W-dag \( W_{b,d} \) by identifying (or, merging) the rightmost
sink of the former dag with the left-
most sink of the latter. W-dags epitomize “expansive” com-
putations.

**M-dags.** For each integer \( d > 1 \), the \((1, d)\)-M-dag \( M_{1,d} \) 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 \( M_{a+b,d} \) is obtained from the \((a, d)\)-M-dag \( M_{a,d} \) and the \((b, d)\)-M-dag \( M_{b,d} \) by identifying (or, merging) the rightmost
source of the former dag with the leftmost
source of the latter. M-dags epitomize “contractive” (or, “reduc-
tive”) computations.

**N-dags.** For each integer \( s > 0 \), the \( s\)-N-dag \( N_s \) 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. Specifically,
\( N_s \) is obtained from \( W_{s-1,2} \) by adding a new source on
the right whose sole arc goes to the rightmost sink. The left-
most source of \( N_s \) has a child that has no other parents; we
call this source the **anchor** of \( N_s \).

**(Bipartite) Cycle-dags.** For each integer \( s > 1 \), the \( s\)-
(Bipartite) Cycle-dag \( C_s \) is obtained from \( N_s \) by adding
a new arc from the rightmost source to the leftmost sink—so
that each source \( v \) has arcs to sinks \( v \) and \( v + 1 \ mod \ s \).

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

We choose the preceding building blocks because the
computation-dags of Fig. 1 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 \((2, 2)\)-W-dag with a \((2, 2)\)-W-dag, then a
\((3, 2)\)-W-dag, then a \((4, 2)\)-W-dag, and so on; the reduction-
line is similarly constructed from its sources upward using
\((k, 2)\)-M-dags for successively decreasing values of \( k \); the
line-tree is constructed from its sources/leaves upward by
“concatenating” independent collections of \((1, 2)\)-M-
dags; the FFT dag is constructed from its sources outward
by “concatenating” independent collections of 2-cycles.

which are identical to 2-cliques).

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 el-
egant 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 mod-
eled by the dag’s arcs—represents the changing (computa-
tional) status of the dag’s task-nodes.

Our study is based on the Internet-Computing (IC, for
short) Pebble Game of [13], whose structure derives from
the “no recomputation allowed” pebble game of [14]. Based
on studies of Internet-based computing in, for instance,
[1, 9, 16], arguments are presented in [13, 15] (q.v.) that
justify studying an idealized, simplified form of the Game.

A. The rules of the Game. The IC Pebble Game on a
computation-dag \( G \) 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 el-
igibility for execution, and EXECUTED pebbles, whose pres-
ence indicates a task’s having been executed. We now
present the rules of the Game, which simplify those of the
original IC Pebble Game of [13, 15].

**The Rules of the Batch-IC Pebble Game**

- **S** begins by placing an ELIGIBLE pebble on each un-
pebbled source of \( G \).
At each step, node has been executed. Indeed, we see in $t$:
\[
\text{such that, at } s \text{is in } t \text{ of the places an } E = \text{schedule \ eventually ELIGIBLE} \]

$E$ replaces that pebble by an EXECUTED pebble, places an ELIGIBLE pebble on each unpebbled node of $G$ all of whose parents contain EXECUTED pebbles.

$S$’s goal is to allocate nodes in such a way that every node $v$ of $G$ eventually contains an EXECUTED pebble.

/*This modest goal is necessitated by the possibility that $G$ is infinite.*/

A schedule for the IC Pebble Game on a dag $G$ 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 (resp., EXECUTED) when it contains an ELIGIBLE (resp., an EXECUTED) pebble. For uniformity, we henceforth talk about executing nodes rather than tasks.

### B. The IC quality of a play of the Game

The goal in the IC Pebble Game is to play the Game in a way that maximizes the production rate of ELIGIBLE pebbles at every step $t$. For each step $t$ of a play of the Game on a dag $G$ under a schedule $\Sigma$: $E_\Sigma(t)$ (resp., $E_\Sigma(t)$) denotes the number of nodes (resp., nonsource nodes) of $G$ that contain ELIGIBLE pebbles at step $t$. (Note that $E_\Sigma(0) = 0$.)

We measure the IC quality of a play of the IC Pebble Game on a dag $G$ by the size of $E_\Sigma(t)$ at each step $t$ of the play—the bigger $E_\Sigma(t)$ is, the better. Our goal is an IC-optimal (ICO, for short) schedule $\Sigma$, in which, for all steps $t$, $E_\Sigma(t)$ 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 $G$ such that, at every step of the computation, $\Sigma$ maximizes the number of ELIGIBLE nodes across all schedules for $G$. 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 3.1.C that there exist dags that do not admit any ICO-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 IC Server receives a batch of requests for nodes at (roughly) the same time, then an IC-optimal schedule ensures that maximally many nodes are ELIGIBLE at that time, so that maximally many requests can be satisfied, thereby exploiting clients’ available resources. See [13, 15] for more elaborate discussions of IC quality.

### 3. On Developing IC-Optimal Schedules

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.B. We then create a framework for scheduling disjoint collections of building blocks and demonstrate the nonexistence of such schedules for certain other collections.

Because sinks produce no ELIGIBLE nodes, while nonsinks may, we can focus on schedules with the following simple structures.

**Lemma 1.** Let $\Sigma$ be a schedule for a dag $G$. If $\Sigma$ is altered to execute all of $G$’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 $G$, Lemma 1 says that we never diminish IC quality by executing all of $G$’s sources before executing any of its sinks.

#### 3.1. ICO Schedules for Sums of Building Blocks

**A. Optimal schedules for individual building blocks.** A schedule for any of the very uniform dags of Fig. 1 is IC-optimal if, and only if, it is parent-oriented—i.e., it executes all parents of a node in consecutive steps [13, 15]. While parent orientation is neither necessary nor sufficient for IC optimality with the “semi-uniform” dags studied here, it is important when scheduling our building-block dags.

**Theorem 1.** Every IC-optimal schedule for a building-block dag is parent-oriented. Parent orientation is also sufficient for IC optimality for all building blocks save N-dags, whose execution must begin with the anchor.

**Proof.** The structures of the building blocks of Section 2.1.B render the following upper bounds on $E_\Sigma(t)$ obvious, as $t$ ranges from 0 to the number of sources in the given dag.2

\[
\begin{align*}
W_{s,d} : & \quad E_\Sigma(t) \leq (d-1)t + |t = s|; \\
N_s : & \quad E_\Sigma(t) \leq t; \\
M_{s,d} : & \quad E_\Sigma(t) \leq \lfloor t = 0 \rfloor + \lceil (t - 1)/(d - 1) \rceil; \\
C_s : & \quad E_\Sigma(t) \leq t - |t = 0| + |t = s|; \\
Q_s : & \quad E_\Sigma(t) = s \times |t = s|.
\end{align*}
\]

---

2 For any statement $P$ about $t$, $[P(t)] = \text{if } P(t) \text{ then } 1 \text{ else } 0$. 

---

0-7695-2312-9/05/$20.00$ (c) 2005 IEEE
The execution orders mandated in the theorem convert each of these inequalities to an equality.

We now develop tools that extend Theorem 1 to disjoint unions—often called sums—of building-block dags. Let \( G_1, \ldots, G_n \) be connected biparite dags that are pairwise disjoint, in that \( N_{G_i} \cap N_{G_j} = \emptyset \) for all distinct \( i \) and \( j \). The sum of these dags, denoted \( G_1 + \cdots + G_n \), is the biparite dag whose node-set and arc-set are, respectively, the unions of the corresponding sets of \( G_1, \ldots, G_n \).

**B. The priority relation.** The following relation on bipartite dags often affords us an easy avenue toward IC-optimal schedules—for complex, as well as bipartite, dags.

Let the disjoint bipartite dags \( G_1 \) and \( G_2 \) have \( s_1 \) and \( s_2 \) sources and admit the IC-optimal schedules \( \Sigma_1 \) and \( \Sigma_2 \), respectively. If the following inequalities hold,\(^3\)

\[
(\forall x \in [0, s_1]) \ (\forall y \in [0, s_2]) : E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \leq E_{\Sigma_1}(\min\{s_1, x + y\}) + E_{\Sigma_2}(\{x + y \} - \min\{s_1, x + y\}).
\]

then we say that \( G_1 \) has priority over \( G_2 \), denoted \( G_1 \triangleright G_2 \).

The inequalities in (1) basically say that one never decreases IC quality by executing a source of \( G_1 \), in preference to a source of \( G_2 \), whenever possible.

It is algorithmically important that \( \triangleright \) is transitive.

**Theorem 2.** The relation \( \triangleright \) on bipartite dags is transitive.

**Sketch.** We can transfer source executions from \( G_3 \) through \( G_2 \) to \( G_1 \) while never reducing the total number of ELIGIBLE sinks. The key complication occurs when the number of unexecuted sources of \( G_1 \) is strictly smaller than the number of sources of \( G_2 \) and the number of executed sources of \( G_3 \), because then a simple transfer leaves some sources of \( G_2 \) EXECUTED. We solve this difficulty by orchestrating our transfers in a nonobvious way.

Theorem 2 has two corollaries that further expose the nature of \( \triangleright \) and that tell us how to schedule any sum of pairwise \( \triangleright \)-comparable building-block dags IC optimally.

**Corollary 1.** Let \( G_1, G_2, \ldots, G_n \) be pairwise disjoint bipartite dags. If \( G_1 \triangleright G_2 \triangleright \cdots \triangleright G_n \), then \( G_1 \triangleright (G_2 + \cdots + G_n) \).

Corollary 1 yields a simple proof of the following.

**Corollary 2.** Let \( G_1, \ldots, G_n \) be pairwise disjoint bipartite dags such that each \( G_i \) admits an IC-optimal schedule \( \Sigma_i \). If \( G_1 \triangleright \cdots \triangleright G_n \), then the schedule \( \Sigma^* \) for the sum \( G_1 + \cdots + G_n \) that executes all sources of \( G_1 \) according to schedule \( \Sigma_1 \), then all sources of \( G_2 \) according to schedule \( \Sigma_2 \), and so on for all \( i \in [1, n] \), is IC optimal.

One can actually prove Corollary 2 without invoking the transitivity of \( \triangleright \), by successively “transferring executions” from each \( G_1 \) to \( G_1-1 \). The algorithm that “implements” the corollary is significantly more efficient if it builds on a proof that uses transitivity. To wit, by exploiting transitivity, one needs “transfer” \( O(n^2) \) executions; without transitivity, one “transfers” \( \Theta(n^3) \) executions in the worst case.

**C. Optimal schedules for sums of building blocks.** We now determine the pairwise priorities among our building-block dags.

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

1. For all \( s \) and \( d \), \( W_{s,d} \triangleright G \) for the following bipartite dags \( G \):
   (a) all \( W \)-dags \( W_{s',d'} \) whenever \( d' < d \), or whenever \( d' = d \) and \( s' \geq s \);
   (b) all \( M \)-dags, \( N \)-dags, and \( C \)-dags;
   (c) Clique-dags \( Q_{s'} \) with \( s' \leq d \).

2. For all \( s, N_s \triangleright G \) for the following bipartite dags \( G \):
   (a) all \( N \)-dags \( N_{s'} \), for all \( s' \); (b) all \( M \)-dags.

3. For all \( s, C_s \triangleright G \) for the following bipartite dags \( G \):
   (a) \( C_s \); (b) all \( M \)-dags.

4. For all \( s \) and \( d \), \( M_{s,d} \triangleright M_{s',d'} \) whenever \( d' > d \), or whenever \( d' = d \) and \( s' \leq s \).

5. For all \( s \), \( Q_s \triangleright Q_{s'} \).

The proof of Theorem 3 is a long sequence of calculations paired with an invocation of the transitivity of \( \triangleright \); we omit it from this extended abstract.

Theorem 3 indicates a sense in which \( M \)-dags are dual to \( W \)-dags: higher degrees (the parameter \( d \)) promote priorities for \( W \)-dags and denote them for \( M \)-dags; increased “lengths” (the parameter \( s \)) denote priorities for \( W \)-dags and promote them for \( M \)-dags.

**D. Incompatible sums of building blocks.** Certain sums of building-block dags do not admit any IC-optimal schedule.

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

1. all sums of the forms \( C_{s_1} + C_{s_2} \) or \( C_{s_1} + Q_{s_2} \) or \( Q_{s_1} + Q_{s_2} \), where \( s_1 \neq s_2 \);
2. all sums of the form \( N_{s_1} + C_{s_2} \) or \( N_{s_1} + Q_{s_2} \);
3. all sums of the form \( Q_{s_1} + M_{s_2,d} \), where \( s_1 > s_2 \).

**Sketch.** All arguments follow the same basic reasoning. Consider, e.g., \( G = C_{s_1} + C_{s_2} \) when \( s_1 \neq s_2 \). The only way to achieve \( E_{\Sigma_1}(s_1) = s_1 \) is for \( \Sigma \) to execute sources of \( C_{s_1} \) for the first \( s_1 \) steps; else, \( E_{\Sigma_1}(s_1) < s_1 \). Similarly, the only way to achieve \( E_{\Sigma_2}(s_2) = s_2 \) is to execute sources of \( C_{s_2} \) for the first \( s_2 \) steps; else, \( E_{\Sigma_2}(s_2) < s_2 \).

We summarize the results of Section 3.1 in Table 1.
4. Compositions of Building-Block Dags

We show now 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.B as a base set.

We inductively introduce dags obtained by composing connected bipartite dags.

- Start with a base set $B$ of connected bipartite dags.
- Given $G_1, G_2 \in B$—which could be copies of the same dag with nodes renamed to achieve disjointness—one obtains a composite dag $G$ as follows.
  - Let $G$ begin as the sum, $G_1 + G_2$. Rename nodes to ensure that $N_G$ is disjoint from $N_{G_1}$ and $N_{G_2}$.
  - Select some set $S_1$ of sinks from the copy of $G_1$ in the sum $G_1 + G_2$, and an equal-size set $S_2$ of sources from the copy of $G_2$.
  - Pairwise identify (i.e., merge) the nodes in $S_1$ and $S_2$ in some way. The resulting set of nodes is $N_G$; the induced set of arcs is $A_G$.
- Add the dag $G$ thus obtained to the base set $B$.

We denote the composition operation by $\uparrow$ and say that the resulting dag $G$ is composite of type $[G_1 \uparrow G_2]$. The roles of $G_1$ and $G_2$ in creating $G$ are asymmetric: $G_1$ contributes sinks, while $G_2$ contributes sources. There is a natural correspondence between the node-set of $G$ and the node-sets of $G_1$ and $G_2$, which we shall exploit.

### Table 1. The relation $\rhd$ among building-block dags. Entries either list conditions for priority or indicate (via “X”) the absence of any IC-optimal schedule for that pairing.

<table>
<thead>
<tr>
<th>$G_1 \rhd G_2$</th>
<th>$W_{s',d'}$</th>
<th>$N_{s'}$</th>
<th>$M_{s',d'}$</th>
<th>$C_{s'}$</th>
<th>$Q_{s'}$</th>
</tr>
</thead>
</table>
| $W_{s,d}$      | $d' < d$   | all $s'$  | all $s', d'$| all $s'$| $s' \leq d$
|                 | $d' = d$   | all $s'$  | all $s', d'$|        | else X |
|                 | $s' \geq s$|           |             |        |        |
| $N_{s}$        | all $s'$   | all $s', d'$|           | X      | X      |
| $M_{s,d}$      | $d' > d$   |           | $d' = d$   | $s' = s$
|                 | $s' \leq s$|           | all $s'$  | else X |
| $C_{s}$        | X           | all $s', d'$| $s' = s$
|                 |             |        | all $s'$  | else X |
| $Q_{s}$        | X           | X for $s > s'$| $s' = s$
|                 |             |        | all $s'$  | else X |

As one would hope, composition is associative, so we do not have to keep track of the order in which dags are incorporated into a composite dag. Fig. 3 illustrates this fact, whose verification is left to the reader.

**Lemma 3.** The composition operation on dags is associative. That is, for all dags $G_1$, $G_2$, $G_3$, a dag $G$ is composite of type $[G_1 \uparrow G_2 \uparrow G_3]$ if, and only if, it is composite of type $[G_1 \uparrow [G_2 \uparrow G_3]]$.

![Figure 3](image-url)
We can now illustrate simply the natural correspondence between the node-set of a composite dag and those of its constituents, via Fig. 1:

- The 2-dimensional mesh is composite of type $\mathcal{W}_{1,2} \uparrow \mathcal{W}_{2,2} \uparrow \mathcal{W}_{3,2} \uparrow \cdots \uparrow \text{ad infinitum}$.
- A binary reduction-tree is obtained by pairwise composing many instances of $\mathcal{M}_{1,2}$ (seven instances in the figure).
- The 5-level 2-dimensional reduction-mesh is 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 $G_2 = Q_2$ (twelve instances in the figure).

Say that dag $G$ is a $\triangleright$-linearization of the connected bipartite dags $G_1, G_2, \ldots, G_n$, if:

1. $G$ is composite of type $\uparrow G_1 \uparrow G_2 \uparrow \cdots \uparrow G_n$;
2. each $G_i \triangleright G_{i+1}$, for all $i \in [1, n] - 1$.

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

**Theorem 4.** Let $G$ be a $\triangleright$-linearization of $G_1, \ldots, G_n$, where each $G_i$ admits an IC-optimal schedule $\Sigma_i$. The schedule $\Sigma$ for $G$ that proceeds as follows is IC optimal.

1. $\Sigma$ executes the nodes of $G_i$ that correspond to sources of $G_1$, in the order mandated by $\Sigma_1$, then the nodes that correspond to sources of $G_2$, in the order mandated by $\Sigma_2$, and so on, for all $i \in [1, n]$.
2. $\Sigma$ finally executes all sinks of $G$ in any order.

**Sketch.** Let $\Sigma'$ be a schedule for $G$ that has maximum $E_{\Sigma'}(t)$ for some $t$, and let $X$ be the first nodes that $\Sigma'$ executed. By Lemma 1, we may assume that $X$ is a subset of the nonsinks of $G$. We then observe that the number of eligible nodes in $G$ is $|S| - |X| + \sum_{i=1}^m e_i(X_i)$, where $S$ is the set of sources of $G$, and $e_i(X_i)$ is the number of sinks of $G_i$ that are eligible when only sources $X_i$ of $G_i$ that correspond to $X$ are executed. Because of the $\triangleright$-priorities among the $G_i$’s, Corollary 2 implies that the sum is maximized for a “low-index-loaded” execution of the $G_i$’s. Since the sources of each $G_j$ could have been merged with sinks of $G_1, \ldots, G_{j-1}$, the execution corresponds to a set $X'$ of $t$ executed nodes of $G$ that satisfy precedence constraints, which ensures that $\Sigma$ is IC optimal.

5. **IC Optimality 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 computation-dag but rather to execute a given one. We now craft a framework that converts the synthetic worldview of Section 4 to an analytical worldview in which a given dag is simplified and decomposed into its constituents. When the simplified dag is composite, i.e., is constructible by composing disjoint bipartite building blocks via composition can be decomposed to expose its constituents and how they combine to yield $G$. 

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 any inter-task dependencies. Often, removing all shortcuts renders a dag composite, hence susceptible to our scheduling strategy.

For any dag $G$ and nodes $u, v \in N_G$, we write $u \leadsto_G v$ to indicate that there is a path from $u$ to $v$ in $G$. An arc $(u \rightarrow v) \in A_G$ is a shortcut if there is a path $u \leadsto_G v$ that does not include the arc. One shows 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) $G'$ of dag $G$ is a subdag that has the same transitive closure [4] as $G$, such that no proper subdag of $G'$ does.

**Lemma 5 ([8]).** Every dag $G$ has a unique transitive skeleton, $\sigma(G)$, which can be found in polynomial time.

**Sketch.** One easily identifies a shortcut, e.g., by removing an arc $(u \rightarrow v)$ and testing if $v$ is still accessible from $u$. Once identified, all shortcuts can be removed.

Since $G$ shares its node-set with $\sigma(G)$, we can use the same schedule to execute both dags. Importantly, any schedule executes $G$ as efficiently (in IC quality) as it executes $\sigma(G)$. A special case of this result appears in [13].

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

**Sketch.** A node becomes eligible in $\sigma(G)$ and $G$ simultaneously, because the dags share their transitive closure.

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

5.2. **Decomposing a Composite Dag**

Every dag $G$ that is constructed from connected bipartite building blocks via composition can be decomposed to expose its constituents and how they combine to yield $G$. 

0-7695-2312-9/05/$20.00 (c) 2005 IEEE
We describe this decomposition process in detail and illustrate it with the dag of Fig. 1 and their constituents.

**Selecting a constituent.** Our process begins by selecting any constituent of \( G \) all of whose sources are also sources of \( G \); call the selected constituent \( B_1 \) (the notation emphasizing that \( B_1 \) is bipartite).

In Fig. 1: Every candidate \( B_1 \) for the FFT dag is a copy of \( C_2 \) included in levels 2 and 3; every candidate for the reduction-tree is a copy of \( M_{1,2} \); the unique candidate for the reduction-mesh is \( M_{4,2} \).

**Detaching a constituent.** One “detaches” \( B_1 \) from \( G \) by deleting the nodes of \( G \) that correspond to sources of \( B_1 \), all incident arcs, and all resulting isolated sinks. One thereby replaces \( G \) with a pair of dags \( \langle B_1, G' \rangle \), where \( G' \) is the remnant of \( G \) remaining after \( B_1 \) is detached. If \( G' \) is not empty, then the process of selection and detachment continues, producing a sequence of the form

\[
G \implies \langle B_1, G' \rangle \implies \langle B_1, \langle B_2, G'' \rangle \rangle \implies \langle B_1, \langle B_2, \langle B_3, G''' \rangle \rangle \rangle \implies \cdots ,
\]

leading ultimately to a complete decomposition of \( G \) into a sequence comprising all of its constituents: \( B_1, B_2, \ldots, B_n \).

We claim that the described process recognizes whether or not \( G \) is composite, and, if so, it produces the constituents from which \( G \) is composed (possibly in a different order from the original composition). The following theorem, which follows by induction, is left to the reader.

**Theorem 6.** Let the dag \( G \) be composite of type \( G_1 \uparrow \cdots \uparrow G_n \). The decomposition process produces a sequence \( B_1, \ldots, B_n \) of constituents of \( G \) such that:

- \( G \) is composite of type \( B_1 \uparrow \cdots \uparrow B_n \);
- \( \{B_1, \ldots, B_n\} = \{G_1, \ldots, G_n\} \).

### 5.3. The Super-Dag Yielded by Decomposing \( G \)

The next step in our strategy is to abstract the structure of \( G \) exposed by its decomposition into \( B_1, \ldots, B_n \) in an algorithmically advantageous way. Thereby, we shift focus from the decomposition to \( G \)'s associated super-dag \( S_G \) which we call a supernode to prevent ambiguity—is one of the \( B_i \)’s. There is an arc in \( S_G \) 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 \( B_i \)’s to produce \( G \). Figs. 4 and 5 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 \( G \) reduces to the question of whether or not \( S_G \) admits a topological sort [4] whose linearization of supernodes is consistent with the relation \( > \). For instance, one derives an IC-optimal schedule for the dag \( G \) of Fig. 3(b) (which is decomposed in Fig. 4) by noting that \( G \) is composite of type \( M_{1,2} \uparrow M_{2,3} \uparrow M_{1,3} \) and that \( M_{1,2} > M_{2,3} \uparrow M_{1,3} \). Indeed, \( G \) points out the challenge in determining if Theorem 4 applies, since it is also composite of type \( M_{2,3} \uparrow M_{1,2} \uparrow M_{1,3} \). We leave to the reader the easy verification that the linearization \( B_1, \ldots, B_n \) is a topological sort of \( S(B_1 \uparrow \cdots \uparrow B_n) \).
5.4. On Using Priorities among Constituents

Our remaining challenge is to determine a topological sort of $S_G$ 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 $G$ is composite of type $B_1 \uparrow \cdots \uparrow B_n$ and that, for each pair of constituents, $B_i, B_j$ with $i \neq j$, either $B_i \triangleright B_j$ or $B_j \triangleright B_i$. Then $G$ is $\triangleright$-linearizable whenever the following holds.

$B_i \triangleright B_j$ whenever $B_j$ is a child of $B_i$ in $S(B_1 \uparrow \cdots \uparrow B_n)$.

Sketch. A $\triangleright$-linearization of $G$ can be obtained by stably sorting any topological sort of $S(B_1 \uparrow \cdots \uparrow B_n)$.

6. Conclusions and Projections

We have developed three notions that we believe form the basis for a theory of scheduling complex computation-dags for Internet-based computing: the priority relation $\triangleright$ on bipartite dags (Section 3.1), the operation of composition of dags (Section 4), and the operation of 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 are optimal in their rate of rendering tasks eligible for allocation to remote clients—whenever such an optimal schedule exists (Theorems 4, 7). We have used our notions to progress beyond the structural uniformity of the building-block dags in [13, 15] to families that are built in structured, yet flexible, ways from a repertoire of uniform building-block dags. The composite dags that we can now schedule optimally encompass not only those studied in [13, 15], but, as illustrated in Fig. 3, also dags that have rather complex structures, including nodes of varying degrees and non-leveled global structure.

Among the directions 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 to deal with building blocks of much more complex structure than those of Section 2.1.B, including less-uniform bipartite dags and non-bipartite 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. This problem could well be co-NP-Complete (because of its underlying universal quantification).

Acknowledgment. A portion of the research of G. Malewicz was done while visiting the TAPADS Group at the Univ. of Massachusetts Amherst. The research of A. Rosenberg and M. Yurkewych was supported in part by NSF Grant CCF-0342417.

References