Advances in IC-Scheduling Theory: Scheduling Expansive and Reductive DAGs and Scheduling DAGs via Duality

Gennaro Cordasco, Grzegorz Malewicz, *Member*, *IEEE*, and Arnold L. Rosenberg, *Fellow*, *IEEE*

Abstract—Earlier work has developed the underpinnings of the *IC-Scheduling Theory*, a framework for scheduling computations having intertask dependencies—modeled via directed acyclic graphs (DAGs)—for Internet-based computing. The goal of the schedules produced is to render tasks eligible for execution at the maximum possible rate, with the dual aim of 1) utilizing remote clients' computational resources well by always having work to allocate to an available client and 2) lessening the likelihood of a computation's stalling for lack of eligible tasks. The DAGs handled by the theory thus far are those that can be decomposed into a given collection of bipartite *building block DAGs* via the operation of *DAG decomposition*. A basic tool in constructing schedules is a relation $\triangleright$, which allows one to "prioritize" the scheduling of a complex DAG's building blocks. The current paper extends the IC-Scheduling Theory in two ways: by expanding significantly the repertoire of DAGs that the Theory can schedule optimally and by allowing one sometimes to shortcut the algorithmic process required to find optimal schedules. The expanded repertoire now allows the theory to schedule optimally, among other DAGs, a large range of DAGs that are either "expansive," in the sense that they grow outward from their sources, or "reductive," in the sense that they grow inward toward their sinks. The algorithmic shortcuts allow one to "read off" an optimal schedule for a DAG from a given optimal schedule for the DAG's *dual*, which is obtained by reversing all arcs (thereby exchanging the roles of sources and sinks).

Index Terms—IC-Scheduling Theory, Internet-based computing, grid computing, global computing, scheduling DAGs, theory.

1 Introduction

Earlier work [16], [18], [19] has developed the underpinnings of the *IC-Scheduling Theory*, a formal framework for studying the problem of scheduling computations having intertask dependencies for the several modalities of Internet-based computing (IC), including grid computing (see [1], [8], and [7]), global computing (see [2]), and Web computing (see [13]). The goal is to craft schedules that maximize the rate at which tasks are rendered eligible for allocation to (hence, for execution by) remote computing workers (henceforth, clients), with the dual aim of 1) enhancing the effective utilization of remote clients by always having work to allocate to an available client and 2) lessening the likelihood of a computation's stalling pending the completion of already allocated tasks. The impetus for this new theory is *temporal unpredictability*: communication with remote clients proceeds over the Internet and, hence, may experience unpredictable delays. Remote clients are not dedicated to remotely allocated work and, hence, may execute that work at an unpredictable rate. These sources of unpredictability make it difficult to accurately identify critical paths in complex computations and, thereby, to successfully apply traditional scheduling techniques. Consequently, a new scheduling paradigm is needed, which acknowledges both the strengths and weaknesses of the Internet as a computational medium.

As a simple but dramatic illustration of the need to pay attention to the rate of rendering tasks eligible for execution, Fig. 1 depicts two strategies for scheduling wave front computations (Fig. 2b). Under the "diagonal shells" strategy (which is optimal [18]), as shown in Fig. 1a, roughly $\sqrt{T}$ tasks are eligible after T task executions. Under the "square shells") strategy, as shown in Fig. 1b, there are never more than three eligible tasks. Thus, during periods when a large number of clients are available, the "diagonal shells" schedule can fruitfully exploit many of them, whereas the "square shells" schedule cannot.

The framework of [16], [18], and [19] idealizes the problem of scheduling computation DAGs[1] for IC via the assumption that tasks are executed in the order of their allocation (this assumption idealizes the hope that the monitoring of clients' past behaviors and current capabilities prescribed in, say, [1], [12], and [20] can render the desired order likely, if not certain). Building on the case studies in [18] and [19], in [16],

1. All technical terms are defined in Section 2.2. For brevity, we henceforth speak of "DAGs" rather than "computation DAGs."

• G. Cordasco is with the Dipartimento di Informatica e Applicazioni, Università di Salerno, via Ponte don Melillo, 84084 Fisciano (SA), Italy. E-mail: cordasco@dia.unisa.it.
• G. Malewicz is with the Department of Engineering, Google, 1600 Amphitheatre Parkway, Mountain View, CA 94043. E-mail: malewicz@google.com.
• A.L. Rosenberg is with the Department of Computer Science, University of Massachusetts, Room 308, Computer Science Building, Department of Computer Science, Amherst, MA 01003-4610. E-mail: rsnbrg@cs.umass.edu.

*Manuscript received 12 July 2006; revised 6 Dec. 2006; accepted 4 Jan. 2007; published online 12 Jan. 2007.
Recommended for acceptance by T. Davis.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TSE-0185-0706.
Digital Object Identifier no. 10.1109/TPDS.2007.1067.*

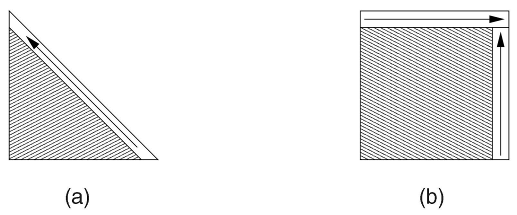


Fig. 1. Scheduling wave front computations by "diagonal" and "square" shells.

we developed what we hope will be the underpinnings of a theory of scheduling complex DAGs for IC. The development in [16] begins with any collection of *building block* DAGs that we know how to schedule optimally. It develops two conceptual/algorithmic notions that allow one to schedule complex DAGs built from these building blocks by using a suite of algorithms that optimally schedule DAGs that arise from a large variety of significant computations:

1. *The* priority *relation* $\triangleright$ *on pairs of DAGs*. The assertion "$\mathcal{G}_1 \triangleright \mathcal{G}_2$" asserts that the schedule Σ that entirely executes $\mathcal{G}_1$ and then entirely executes $\mathcal{G}_2$ is at least as good (relative to our quality metric) as any other schedule that executes both $\mathcal{G}_1$ and $\mathcal{G}_2$.
2. *The operation of* composition *on pairs of DAGs*. This builds up simple DAGs into complex ones.

The suite of algorithms developed in [16] decompose a given DAG $\mathcal{G}$ into building blocks that compose to produce $\mathcal{G}$. One can often use $\triangleright$-priorities among $\mathcal{G}$'s building blocks to compute an optimal schedule for $\mathcal{G}$ from optimal schedules for its building blocks.

Although many DAGs do not admit optimal schedules under IC-Scheduling Theory,[2] the DAGs that arise from a large variety of disparate important computations can be scheduled optimally via the algorithms in [16]. Fig. 2 depicts four such DAGs whose optimal schedules are derived in [5], using the algorithmic framework in [16]. Techniques from the current paper greatly streamline the derivations of these optimal schedules. More importantly, the current paper extends the framework in [16] in two significant ways:

1. In Section 3, we significantly augment the repertoire of building block DAGs that the scheduling algorithms in [16] can accommodate when seeking optimal schedules. This augmentation allows the theory to schedule optimally, *inter alia*, a large range of DAGs that are either "expansive," in the sense that they grow outward from their sources, or "reductive," in the sense that they grown inward toward their sinks. Some structurally uniform such DAGs are exemplified in Fig. 2. The algorithms can also now handle a large range of compositions of "expansive" DAGs with "reductive" DAGs, as exemplified in Fig. 3.
2. In Section 4, we show how we can "read off" two algorithmically significant features of a DAG $\mathcal{G}$ from analogous features for $\mathcal{G}$'s *dual* DAG $\widetilde{\mathcal{G}}$, which is obtained by reversing all of $\mathcal{G}$'s arcs:
   - We show how we can "read off" an optimal schedule for $\mathcal{G}$ from a given optimal schedule for $\widetilde{\mathcal{G}}$, if one exists (Section 4.1). One can, for example, "read off" an optimal schedule for either an "expansive" or a "reductive" DAG from a given optimal schedule for the other.
   - We show how we can "read off" $\triangleright$-priorities among DAGs from known $\triangleright$-priorities among their duals (Section 4.2).

2. Our optimal schedules must maximize the number of eligible nodes *at every step of the computation*, so many DAGs do not admit optimal schedules [16]. For such DAGs, the initial task executions that maximize the number of eligible tasks at time t_1 differ from those that maximize the number at some $t_2 > t_1$.

Thus, in this paper, we expand the repertoire of DAGs that the nascent theory in [16] can schedule optimally, and we provide tools that often allow one to avoid the low-degree polynomial, yet not insignificant, suite of algorithms used in [16] to find optimal schedules.

Related work. Most closely related to our study are its companions in developing IC-Scheduling Theory:

- Reference [16], whose contributions we have just described,
- References [18] and [19], which are case studies that characterize and specify optimal schedules for a variety of uniform DAGs,
- Reference [4], wherein the algorithmic framework in [16] is extended in several ways,
- Reference [5], wherein optimal algorithms are designed for a range of disparate computational problems, including those in Fig. 2,
- References [10] and [14], which evaluate the computational impact of our idealized scheduling theory via simulation experiments and suggest that this impact is quite positive for a substantial range of plausible scenarios,
- Reference [15], which is motivated by the fact that many DAGs do not admit an optimal schedule in the sense of [16] and pursues an orthogonal regimen for scheduling DAGs for IC, 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 the batched framework, but achieving it may entail a prohibitively complex computation), and
- Reference [9], which presents a probabilistic approach to the problem of executing tasks on unreliable clients.

Finally, the impetus for our study derives from the many exciting systems-oriented and/or application-oriented studies of IC, in sources such as [1], [2], [7], [8], [12], [13], and [20].

2 A Basis for a Scheduling Theory

2.1 Computation DAGs

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 of 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:

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

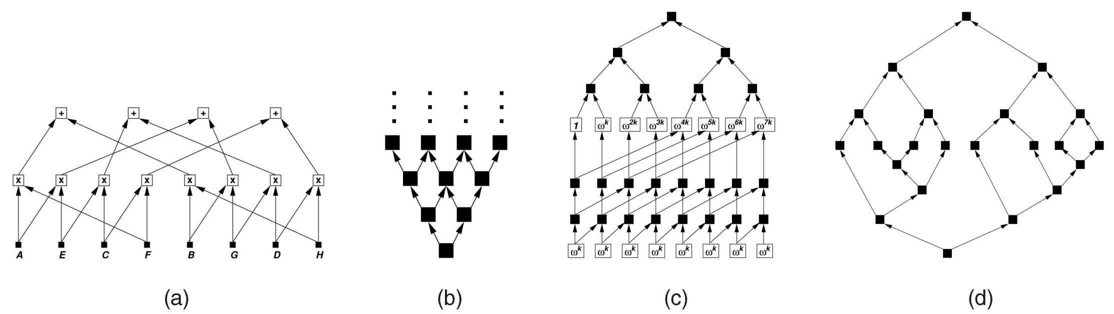


Fig. 2. Data-dependency DAGs for four computations. (a) Matrix multiplication. (b) A wave front computation. (c) The discrete Laplace transform. (d) A divide-and-conquer computation.

A *DAG* $\mathcal{G}$ is a directed graph that has no cycles so that no path of the preceding form has $u_1 = u_n$. When a DAG $\mathcal{G}$ is used to model a computation, that is, is a *computation DAG* (henceforth, *DAG*),

- each node $v \in N_{\mathcal{G}}$ represents a task in the computation and
- 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.

For any arc $(u \rightarrow v) \in A_{\mathcal{G}}$, u is a *parent* of v, and v is a *child* of u in $\mathcal{G}$. The *indegree* (respectively, *outdegree*) of node u is its number of parents (respectively, children). A parentless node of $\mathcal{G}$ is a *source*, and a childless node is a *sink*. $\mathcal{G}$ is *connected* if, when one ignores the orientation of its arcs, there is a path connecting every pair of distinct nodes. $\mathcal{G}$ is *bipartite* if $N_{\mathcal{G}}$ consists entirely of sources and sinks.[3] We refer to a connected bipartite DAG as a *Connected Bipartite Building Block (CBBB)*, a term whose origins will become clear later.

Let the DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$ be *disjoint* in the sense that $N_{\mathcal{G}_1} \cap N_{\mathcal{G}_2} = \emptyset$. The *sum* (or disjoint union) of $\mathcal{G}_1$ and $\mathcal{G}_2$, denoted $\mathcal{G}_1 + \mathcal{G}_2$, is the DAG whose node set is $N_{\mathcal{G}_1} \cup N_{\mathcal{G}_2}$ and whose arc set is $A_{\mathcal{G}_1} \cup A_{\mathcal{G}_2}$.

2.2 A Model for Executing DAGs on the Internet

"Pebble games" on DAGs have yielded elegant formalizations of a variety of problems related to scheduling DAGs. Such games use tokens or *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 *IC Pebble Game* of [18]. Based on the studies of IC in, for example, [1], [12], and [20], arguments are presented in [18] and [19] (which see) that justify the simplified form of the game that we study.

2.2.1 The Rules of the Game.

The IC Pebble Game on a 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. The game is played as follows:

3. Perforce, all arcs go from a source to a sink.

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.*/

A *schedule* for the IC Pebble Game 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, EXECUTED) pebble. For uniformity, we henceforth talk about executing nodes rather than tasks.

2.2.2 The Quality of a Play of the Game.

Our goal is to play the IC Pebble Game in a way that maximizes the production rate of ELIGIBLE pebbles. When $\mathcal{G}$ is bipartite, it suffices to focus on maximizing the production rate of ELIGIBLE *sinks*. For each step t of a play of the Game on a DAG $\mathcal{G}$ under a schedule Σ, let $E_{\Sigma}(t)$ denote the number of ELIGIBLE pebbles on $\mathcal{G}$'s *nonsource* nodes at step t.

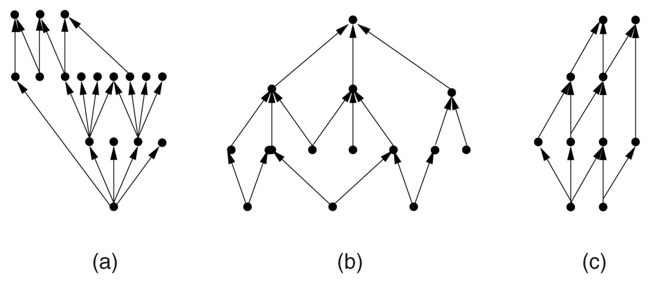


Fig. 3. Three composite DAGs that our expanded framework can schedule optimally. Only (b) could be scheduled optimally by using the framework in [16].

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

The significance of IC quality—hence, of IC optimality—stems from the following scenarios: 1) Schedules that produce ELIGIBLE nodes more quickly may reduce the chance of the "gridlock" that could occur when remote clients are slow so that new tasks cannot be allocated, pending the return of already allocated ones, and 2) if the IC Server receives a batch of requests for tasks at (roughly) the same time, then having more ELIGIBLE tasks available allows the Server to satisfy more requests.

Although our scheduling model is very idealized, preliminary simulation experiments in [14] and [10] suggest that schedules with greater IC quality actually decrease the computation completion times in a large range of IC scenarios.

2.3 A Framework for Crafting IC-Optimal Schedules

The priority relation $\triangleright$. For $i = 1, 2$, let the DAG $\mathcal{G}_i$ have n_i nonsinks and let it admit the IC-optimal schedule Σ_i. If the following inequalities hold,4

$$\begin{aligned}&(\forall x \in [0, n_1])\ (\forall y \in [0, n_2]) :\\ &\quad E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \ \leq \\ &\quad E_{\Sigma_1}(\min\{n_1, x+y\}) + E_{\Sigma_2}(\max\{0, x+y-n_1\}),\end{aligned} \tag{1}$$

then $\mathcal{G}_1$ *has priority over* $\mathcal{G}_2$, denoted $\mathcal{G}_1 \triangleright \mathcal{G}_2$. Informally, one never decreases IC quality by executing a nonsink of $\mathcal{G}_1$ whenever possible.

A framework for scheduling complex DAGs. The operation of *composition* is defined inductively as follows:

- Start with a set $\mathcal{B}$ of base DAGs.5
- One composes DAGs $\mathcal{G}_1$, $\mathcal{G}_2 \in \mathcal{B}$—which could be the same DAG with nodes renamed to achieve disjointness—to obtain a composite DAG $\mathcal{G}$ as follows:
  - Let $\mathcal{G}$ begin as the sum $\mathcal{G}_1 + \mathcal{G}_2$, with nodes renamed if necessary to ensure that $N_\mathcal{G} \cap (N_{\mathcal{G}_1} \cup N_{\mathcal{G}_2}) = \emptyset$.
  - 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-sized set S_2 of sources from the copy of $\mathcal{G}_2$ in the sum.
  - Pairwise identify (that is, merge) the nodes in the sets S_1 and S_2 in some way. The resulting set of nodes is $\mathcal{G}$'s node set, and the induced set of arcs is $\mathcal{G}$'s arc set.6
- Add the DAG $\mathcal{G}$ thus obtained to the base set $\mathcal{B}$.

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

The DAG $\mathcal{G}$ is a $\triangleright$*-linear composition* of the CBBBs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ if 1) $\mathcal{G}$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$ and 2) each $\mathcal{G}_i \triangleright \mathcal{G}_{i+1}$ for all $i \in [1, n-1]$.

The following results underlie our decomposition-based scheduling algorithm.

4. $[a, b]$ denotes the set of integers $\{a, a+1, \ldots, b\}$.
5. Continuing the practice in [16], our base DAGs here are CBBBs.
6. An arc $(u \rightarrow v)$ is *induced* if $\{u, v\} \subseteq N_\mathcal{G}$.

Lemma 2.1 ([16]). ***(a)*** *If a schedule Σ for a DAG $\mathcal{G}$ is altered to execute all of $\mathcal{G}$'s nonsinks before any of its sinks, then IC quality of the resulting schedule is no less than Σ's.* ***(b)*** *The relation $\triangleright$ is transitive.* ***(c)*** *The composition operation is associative.*

Theorem 2.1 ([16]). *Let $\mathcal{G}$ be a $\triangleright$-linear composition of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, where each $\mathcal{G}_i$ admits an IC-optimal schedule Σ_i. The schedule Σ for $\mathcal{G}$ that proceeds as follows is IC optimal:*

1. *For $i = 1, \ldots, n$, in turn, Σ executes the nodes of $\mathcal{G}$ that correspond to the nonsinks of $\mathcal{G}_i$ in the order mandated by Σ_i.*
2. *Σ finally executes all the sinks of $\mathcal{G}$ in any order.*

One finds in [16] a suite of algorithms that determine whether or not a given DAG $\mathcal{G}$ can be decomposed into a set of CBBBs $\{\mathcal{G}_i\}$ that satisfy Theorem 2.1: They compute an IC-optimal schedule for $\mathcal{G}$ whenever the theorem applies. We summarize the algorithms to suggest their low-degree polynomial yet superlinear complexity:

1. Remove from $\mathcal{G}$ all "shortcuts," that is, arcs whose removal does not alter connectivity (see [11]).
2. Parse $\mathcal{G}$, when possible, into CBBBs via a greedy algorithm.
3. Use the parsing to transform $\mathcal{G}$ into a "super-DAG" of CBBBs whose arcs denote compositions.
4. Use a stable sorting algorithm [6] to determine if $\mathcal{G}$ is a $\triangleright$-linear composition of its CBBBs (so that Theorem 2.1 yields an IC-optimal schedule for $\mathcal{G}$).

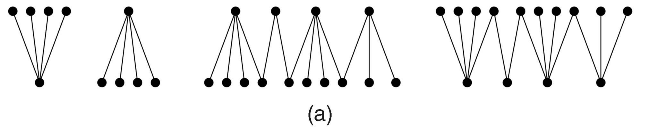
The nascent scheduling theory in [16] is illustrated there via a small repertoire of CBBBs that lead, via Theorem 2.1, to a rich family of complex DAGs, which we know how to schedule IC optimally (including the DAGs in Fig. 2). This early success motivates the challenge that we address in Section 3: to expand the repertoire of CBBBs that the theory can handle. The complexity of the scheduling algorithms derived in [16] from Theorem 2.1 motivates the challenge that we address in Section 4: to identify situations in which we can bypass those algorithms by having a DAG "inherit" an IC-optimal algorithm from some kindred DAG.

3 Expanding the Repertoire of Building Blocks

This section is devoted to expanding the repertoire of CBBBs that the scheduling algorithms in Section 2.3 have access to in [16]. We focus only on the problem of finding IC-optimal schedules for the CBBBs that we consider, relying on the algorithm developed in [4] to find all possible $\triangleright$-priorities among these DAGs.

3.1 Planar Bipartite Trees

We strive for an extensive repertoire of CBBBs 1) that compose into DAGs that one might encounter in real computations and 2) that our theory shows how to schedule optimally and prioritize. Although our main focus is on DAGs that are either *expansive* (growing outward from their sources) or *reductive* (growing inward toward their sinks; see Figs. 2 and 4), we gain a technical advantage by considering also CBBBs that combine expansive and reductive behavior. Specifically, by demonstrating how we can schedule any such combined CBBB IC optimally, we demonstrate in one fell swoop how we can


(a)

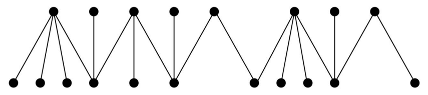

(b)

Fig. 4. (a) $\mathcal{W}[4]$, $\mathcal{M}[4]$, $\mathcal{M}[4,2,4,3]$, $\mathcal{W}[4,2,4,3]$. (b) $\mathcal{P}[\frac{1}{4},3,\frac{1}{3},3,\frac{1}{2},2,\frac{1}{4},3,\frac{1}{2}]$. Edges represent arcs that point upward.

schedule IC-optimally any CBBB that is either expansive or reductive.

Planar Bipartite Trees. A sequence of positive numbers is *zigzagged* if it alternates integers and reciprocals of integers, with all integers exceeding 1. For any zigzagged sequence $\hat{\delta}$, the $\hat{\delta}$ **PBT**, denoted $\mathcal{P}[\hat{\delta}]$, is defined inductively as follows (see Fig. 4).

The base cases. For each $d > 1$:

- $\mathcal{P}[d]$ is the (single-source) *outdegree-d* **W-DAG** $\mathcal{W}[d]$, that is, the bipartite DAG that has one source, d sinks, and d arcs connecting the source to each sink.
- $\mathcal{P}[\frac{1}{d}]$ is the (single-sink) *indegree-d* **M-DAG** $\mathcal{M}[d]$, that is, the bipartite DAG that has one sink, d sources, and d arcs connecting each source to the sink.

The inductive extensions. For each zigzagged sequence $\hat{\delta}$ and each $d > 1$,

- if $\hat{\delta}$ ends with a reciprocal, then $\mathcal{P}[\hat{\delta}, d]$ is obtained by giving $d-1$ new sinks to $\mathcal{P}[\hat{\delta}]$, with $\mathcal{P}[\hat{\delta}]$'s rightmost source as their common parent, and
- if $\hat{\delta}$ ends with an integer, then $\mathcal{P}[\hat{\delta}, \frac{1}{d}]$ is obtained by giving $d-1$ new sources to $\mathcal{P}[\hat{\delta}]$, with $\mathcal{P}[\hat{\delta}]$'s rightmost sink as their common child.

Special PBTs. Two classes of PBTs deserve special mention. For any sequence of integers, $\hat{\delta} = d_1, d_2, \ldots, d_k$, each > 1, the **$\hat{\delta}$-strand of W-DAGS**, denoted $\mathcal{W}[\hat{\delta}]$, and the **$\hat{\delta}$-strand of M-DAGs**, denoted $\mathcal{M}[\hat{\delta}]$ are defined as follows:

$$\mathcal{W}[d_1, d_2, \ldots, d_k] = \mathcal{P}\left[d_1, \frac{1}{2}, d_2, \frac{1}{2}, \ldots, \frac{1}{2}, d_k\right],$$
$$\mathcal{M}[d_1, d_2, \ldots, d_k] = \mathcal{P}\left[\frac{1}{d_1}, 2, \frac{1}{d_2}, 2, \ldots, 2, \frac{1}{d_k}\right].$$

We refer generically to $\mathcal{M}[\hat{\delta}]$ as an **M-strand** and to $\mathcal{W}[\hat{\delta}]$ as a **W-strand**.[7] Note that every strand is connected and hence has no isolated nodes.

Fig. 2 illustrates why we view W-strands and M-strands as the basic building blocks of expansive and reductive DAGs, respectively. The expansive mesh is a composite of type $\mathcal{W}[2] \Uparrow \mathcal{W}[2,2] \Uparrow \mathcal{W}[2,2,2]$; the expansive tree is composed of seven instances of $\mathcal{W}[2]$. Dually, the reductive mesh is composite of type $\mathcal{M}[2,2,2] \Uparrow \mathcal{M}[2,2] \Uparrow \mathcal{M}[2]$; the reductive tree is composed of seven instances of $\mathcal{M}[2]$. Fig. 5 illustrates this for the reductive mesh.

7. In [16], W-strands of the form $\mathcal{W}[2,2,\ldots,2,1]$ are called **N-strands**.

3.2 IC-Optimal Schedules for M-Strands, W-Strands, and PBTs

Theorem 3.1. *Every sum of PBTs admits an IC-optimal schedule.*

Because M-strands and W-strands are special forms of PBTs, Theorem 3.1 immediately implies that *every sum of M-Strands or W-strands admits an IC-optimal schedule*.

Our proof of Theorem 3.1, in fact, proves a stronger assertion, which we develop now.

Focus on an arbitrary sum of PBTs $\mathcal{S}$. By Lemma 2.1a, we may restrict attention to schedules that execute all of $\mathcal{S}$'s sources before any of its sinks. Let $Src(\mathcal{S})$ denote the set of $\mathcal{S}$'s sources. For any $X \subseteq Src(\mathcal{S})$, denote by $e(X;\mathcal{S})$ the number of sinks of $\mathcal{S}$ that are ELIGIBLE at the step when the sources in X are precisely those that have been executed.

Focus on a planar drawing of $\mathcal{S}$ strand by strand so that we can identify its sources, from left to right, with the integers $1, 2, \ldots, n$, where $n = |Src(\mathcal{S})|$. For any $u \in Src(\mathcal{S})$ and any $k \in [1,n]$, denote by $u[k]$ the set $u[k] = \{u, u+1, \ldots, u+k-1\}$ comprising source u and the $k-1$ sources to its right in the drawing. To simplify exposition, we allow k to exceed $n+1-u$ so that $u[k]$ may contain numbers greater than n, *but none of these large integers denotes a node of $\mathcal{S}$*. Let $\mathcal{S}_u$ denote the strand of $\mathcal{S}$ that u belongs to and let $S_u = Src(\mathcal{S}_u)$. Denote by $e_k(u)$ the number $e_k(u) = e(u[k] \cap S_u; \mathcal{S})$ of the sinks of $\mathcal{S}$ that are rendered ELIGIBLE when the sources in $u[k]$ are the only EXECUTED sources of $\mathcal{S}$. Associate with each $u \in Src(\mathcal{S})$ the n-component *eligiblity vector* $V_u = \langle e_1(u), e_2(u), \ldots, e_n(u)\rangle$. Note that, since $|Src(\mathcal{S})| = n$, if $u[k]$ contains numbers larger than n, then the last u entries of V_u are identical. Finally, order the vectors $V_1, V_2, \ldots, V_n$ *in lexicographic order* by using the notation $V_v \geq_L V_w$ to indicate that source v's vector precedes source w's vector lexicographically. We call a source $s \in Src(\mathcal{S})$ *maximal* if $V_s \geq_L V_{s'}$ for all $s' \in Src(\mathcal{S})$.

A schedule $\Sigma_\mathcal{S}$ for the sum of PBTs $\mathcal{S}$ is *$\geq_L$-greedy* if it operates as follows:

1. $\Sigma_\mathcal{S}$ chooses any maximal $s \in Src(\mathcal{S})$ as the first source of $\mathcal{S}$ to execute.
2. After executing source s, $\Sigma_\mathcal{S}$ removes from $\mathcal{S}$ source s and all sinks (if any) that have s as their only parent. This converts $\mathcal{S}$ to a new sum of PBTs $\mathcal{S}'$ whose constituent strands are given as follows:
   - Each PBT of $\mathcal{S}$ other than $\mathcal{S}_s$ (that is, each strand that does not contain s) is a constituent strand of $\mathcal{S}'$.
   - $\mathcal{S}_s$ contributes to $\mathcal{S}'$ all of its nodes other than s and all sinks that have s as their only parent.

   The number of constituent strands of $\mathcal{S}'$ is one smaller than the number of constituent strands of $\mathcal{S}$ (if s is $\mathcal{S}_s$'s only source), is the same as that number (if s is either the leftmost or the rightmost source of $\mathcal{S}_s$), or is one greater than that number (in all other cases).
3. $\Sigma_\mathcal{S}$ recursively executes the sum $\mathcal{S}'$ by using the $\geq_L$-greedy schedule $\Sigma_{\mathcal{S}'}$.

We prove Theorem 3.1 via the following more detailed result.

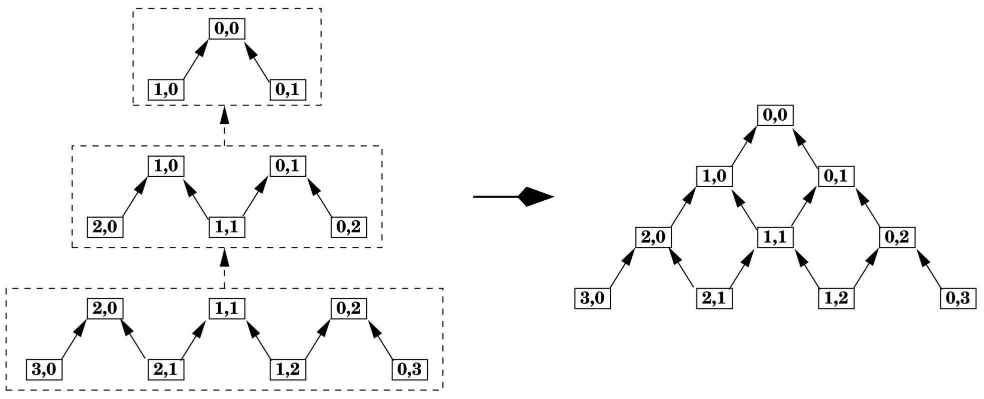


Fig. 5. The reductive mesh of Fig. 2 is composite of type $\mathcal{M}[2,2,2] \Uparrow \mathcal{M}[2,2] \Uparrow \mathcal{M}[2]$.

Theorem 3.2. *For any sum of PBTs $\mathcal{S}$, every $\geq_L$-greedy schedule $\Sigma_\mathcal{S}$ is IC optimal.*

(Intuitively, the "lookahead" inherent in eligibility vectors prevents such a $\Sigma_\mathcal{S}$ from "getting stuck in" local optima that are not globally optimal.) We prove Theorem 3.2 by induction in the next two sections.

3.2.1 $\Sigma_\mathcal{S}$ Chooses the Best First Source to Execute

We show in this section that any $\geq_L$-greedy schedule starts out by executing a source of $\mathcal{S}$ that is most advantageous (with respect to IC quality).

Lemma 3.1. *Let $\mathcal{S}$ be an arbitrary sum of PBTs and let s be an arbitrary maximal source of $\mathcal{S}$. For each set $X \subseteq Src(\mathcal{S})$, there is a set $X' \subseteq Src(\mathcal{S})$ of equal cardinality such that 1) $s \in X'$ and 2) $e(X';\mathcal{S}) \geq e(X;\mathcal{S})$.*

Proof. For any maximal source s of $\mathcal{S}$, focus on an arbitrary $X \subseteq Src(\mathcal{S})$ that does not contain s. Let $w \in X$ be any source and $\ell \in [1, n]$ be any integer such that $w[\ell]$ is a maximal cardinality sequence of consecutive numbers from $X \cap S_w$ (using the integer names of sources). The maximality of source s ensures that $V_s \geq_L V_w$; that is, either $V_s = V_w$ or there exists $h \in [0, n-1]$ such that $s(1) = w(1)$, $s(2) = w(2), \ldots, s(h) = w(h)$, $s(h+1) > w(h+1)$.

We now investigate the impact on the number of ELIGIBLE sinks at step $|X|$ of the execution of $\mathcal{S}$ and of "unexecuting" certain sources from set X and executing an equal number of other sources in their stead. Specifically, we "unexecute" some EXECUTED sources near to and including w and execute an equal number of un-EXECUTED sources near to and including s (this substitution for certain nodes in X will yield the set X').

Focus on the following two quantities:

- $h^\star \in [0, n-1]$ is the largest value such that $e_1(s) = e_1(w)$, $e_2(s) = e_2(w), \ldots, e_{h^\star}(s) = e_{h^\star}(w)$. It is thus the maximum number of consecutive sources starting from s whose execution *does not* improve on the "yield" produced by starting from w.
- $k^\star \in [1, \ell]$ is the largest value such that $s[k^\star] \cap X = \emptyset$. It is thus the maximum number of consecutive sources starting from s that are *not* EXECUTED at the start of our modification of X.

We branch on the relative sizes of $h^\star$ and $k^\star$.

Case 1: $h^\star < k^\star$. Consider the impact of "unexecuting" the sources in $w[h^\star + 1]$ and executing, in their steads, the sources in $s[h^\star + 1]$ (which, in this case, is a subset of S_s). This replaces the set X by $X' = (X \setminus w[h^\star + 1]) \cup s[h^\star + 1]$. The "unexecution" loses us at most $e_{h^\star+1}(w) + 1$ ELIGIBLE sinks (the possible extra sink may be lost if the rightmost source in $w[h^\star + 1]$ has a right-hand neighbor in $\mathcal{S}_w \cap X$ with which it shares a child). In compensation, the execution gains us at least $e_{h^\star+1}(s) \geq e_{h^\star+1}(w) + 1$ new ELIGIBLE sinks. We are thus not worse off for the exchange.

Case 2: $h^\star \geq k^\star$. We have two subcases to consider:

Subcase 2.1: $e_{k^\star}(w) = e_\ell(w)$.

In this case, we know two significant pieces of information:

1. Once having executed nodes w, $w+1, \ldots,$ $w + k^\star - 1$, no further sinks were rendered ELIGIBLE by executing any source in $\{w + k^\star, \ldots, w + \ell - 1\}$ (recall that $w[\ell] \subseteq X$).
2. All of nodes s, $s+1, \ldots, s + k^\star - 1$ belong to S_s, and none belong to X.

This information tells us that "unexecuting" the sources in $w[k^\star]$ loses us *exactly* $e_{k^\star}(w)$ ELIGIBLE sinks, whereas executing the sources in $s[k^\star]$ gains us at least $s_{k^\star}(w) = e_{k^\star}(w)$ new ELIGIBLE sinks. This means that replacing the set X by $X' = (X \setminus w[k^\star]) \cup s[k^\star]$ leaves us with at least as many ELIGIBLE sinks. As in Case 1, we are not worse off for the exchange.

Subcase 2.2: $e_{k^\star}(w) < e_\ell(w)$.

In this case, even having executed nodes $w, w+1, \ldots,$ $w + k^\star - 1$, some new sinks *are* rendered ELIGIBLE by executing the sources in $\{w + k^\star, \ldots, w + \ell - 1\}$. This means that $s + k^\star \in S_s$; that is, the integer $s + k^\star$ is the name of a real source of $\mathcal{S}$. To wit, were this not true, we would have $e_{k^\star+1}(s) = e_{k^\star}(s)$ for all $i > 1$ so that, in particular, $e_\ell(s) < e_\ell(w)$. This would contradict the fact that $V_s \geq_L V_w$.

Let t be the leftmost child of $s + k^\star$ (using the planar drawing of $\mathcal{S}$ to specify "leftmost"). Since source $s + k^\star - 1 \notin X$, t was not rendered ELIGIBLE by the execution of the sources in X. Furthermore, since $s + k^\star \notin s[k^\star]$, t will not be rendered ELIGIBLE by the execution of the sources in $s[k^\star]$.

We digress to introduce a useful notion. A source (respectively, sink) u of a PBT is a *backbone* source (respectively, sink) if u is either extremal in the strand (that is, leftmost or rightmost) or u has at least two children (respectively, at least two parents). We note the obvious yet important fact that *every maximal source is a backbone source*. To wit, at least one of the two neighboring (in the planar drawing) backbone sources of each nonbackbone source s has a lexicographically larger eligibility vector than s.

Our analysis now branches on whether or not $s + k^\star$ is a backbone source of $\mathcal{S}$.

Subcase 2.2.1. If $s + k^\star$ is a backbone source, then we know four significant facts:

1. $s + k^\star \in X$ because $h^\star \geq k^\star$.
2. $s + k^\star$ is a backbone source by assumption.
3. s is a backbone source because it is maximal.
4. $s + k^\star \notin w[k^\star]$ because $V_s \geq_L V_w$ and $h^\star \geq k^\star$.

It follows that executing the nodes in $X \cup s[k^\star]$ renders sink t ELIGIBLE. Therefore, "unexecuting" the sources in $w[k^\star]$ loses us at most $e_{k^\star}(w) + 1$ ELIGIBLE sinks. In compensation, executing the sources in $s[k^\star]$ gains us at least $e_{k^\star}(s) + 1$ ELIGIBLE sinks because t becomes ELIGIBLE when source $s + k^\star - 1$ is executed. Thus, replacing X with $X' = (X \setminus w[k^\star]) \cup s[k^\star]$ leaves us with at least as many ELIGIBLE sinks.

Subcase 2.2.2. If $s + k^\star$ is not a backbone source, then in order to render sink t ELIGIBLE, we must execute not only $s + k^\star$ but also the set $R \subset S_s$ that comprises all of the sources to $s + k^\star$'s right (in the drawing) until we reach the next backbone source.

Let $j^\star \in [0, n]$ be the largest integer such that $e_{k^\star}(w) = e_{k^\star + j^\star}(w)$; that is,

$$e_{k^\star}(w) = e_{k^\star+1}(w) = e_{k^\star+2}(w) = e_{k^\star+j^\star}(w) < e_{k^\star+j^\star+1}(w).$$

Since the current case is defined by the condition "$e_{k^\star}(w) < e_\ell(w)$," we know that *every element of* $w[k^\star + j^\star]$ *belongs to* X.

Easily, $|R| \leq j^\star$. Were this not the case, we would have

$$e_{k^\star+j^\star+1}(s) = e_{k^\star+j^\star}(s) = \cdots = e_{k^\star}(s) = e_{k^\star}(w) < e_{k^\star+j^\star+1}(w).$$

This would contradict the fact that $V_s \geq_L V_w$.

Note now that "unexecuting" the $k^\star + j^\star$ sources in $w[k^\star + j^\star]$ would lose us no more than $e_{k^\star}(w) + 1$ ELIGIBLE sinks. In compensation, executing the $k^\star$ sources in $s[k^\star]$ plus the at most $j^\star$ sources in R gains us at least $e_{k^\star}(s) + 1$ ELIGIBLE sinks (because sink t becomes ELIGIBLE). Thus, replacing the set X with the (possibly smaller) set $X' = (X \setminus w[k^\star + j^\star]) \cup (s[k^\star] \cup R)$ leaves us with at least as many ELIGIBLE sinks.

The sequences of cases that we have considered have established the following: By executing a sequence of sources starting with s, instead of a like-numbered sequence starting with w, we can only increase the total number of ELIGIBLE sinks. The lemma follows. □

3.2.2 $\Sigma_{\mathcal{S}}$ Continues to Make Good Choices

Assume for induction that every $\geq_L$-greedy schedule is IC optimal for all sums of PBTs having n or fewer sources. Lemma 3.1 verifies the $(n = 1)$ case of this assertion. Focus, therefore, on an arbitrary sum of PBTs having $n + 1$ sources. Lemma 3.1 shows that every $\geq_L$-greedy schedule $\Sigma_{\mathcal{S}}$ chooses an optimal source $s \in Src(\mathcal{S})$ to execute in its first step. Hence, if we seek a set $X \subseteq Src(\mathcal{S})$ that renders maximally many sinks of $\mathcal{S}$ ELIGIBLE among cardinality-$|X|$ subsets of $Src(\mathcal{S})$, then we can, with no loss of generality, choose X to contain s. Let $\mathcal{S}'$ be the sum of PBTs obtained by removing from $\mathcal{S}$ source s plus all sinks that have s as their only parent.

Let the *yield* $Yld(v)$ of a source v be the number of sinks that are rendered ELIGIBLE by executing just v. Removing s and its single-parent children effectively increases by 1 the yields of the sources that neighbor s in $\mathcal{S}_s$, if any.

Easily, $e(X; \mathcal{S}) = Yld(s) + e(X \setminus \{s\}; \mathcal{S}')$. Since $\mathcal{S}'$ has fewer sources than $\mathcal{S}$, our inductive assumption asserts that schedule $\Sigma_{\mathcal{S}'}$ is IC optimal for $\mathcal{S}'$. It follows that schedule $\Sigma_{\mathcal{S}}$, which executes s and then mimics schedule $\Sigma_{\mathcal{S}'}$, is IC optimal for $\mathcal{S}$.

This completes the proof of Theorem 3.2 and, hence, of Theorem 3.1.

4 Exploiting Duality When Scheduling DAGs

The results in this section apply to *arbitrary* DAGs, not just bipartite ones. Indeed, these results represent a significant step in liberating our scheduling theory from its focus on DAGs that are obtained via composition from any particular repertoire of CBBBs.

The *dual* of a DAG $\mathcal{G}$ is the DAG $\widetilde{\mathcal{G}}$ that is obtained by reversing all of $\mathcal{G}$'s arcs. Clearly, the sources of $\mathcal{G}$ are the sinks of $\widetilde{\mathcal{G}}$, and the sinks of $\mathcal{G}$ are the sources of $\widetilde{\mathcal{G}}$.

Note that the dual of a PBT is another PBT, the dual of an M-strand is a W-strand, and the dual of a W-strand is an M-strand. Specifically, for all alternations of integers and reciprocals,

$$\widetilde{\mathcal{P}}[d_1, d_2, \ldots, d_n] = \mathcal{P}\left[\frac{1}{d_1}, \frac{1}{d_2}, \ldots, \frac{1}{d_n}\right].$$

In this section, we demonstrate that, for any DAG $\mathcal{G}$,

- one can easily "read off" an IC-optimal schedule for either $\mathcal{G}$ or $\widetilde{\mathcal{G}}$ from an IC-optimal schedule for the other (Section 4.1), and
- one can easily determine $\triangleright$-priority relationships of either $\mathcal{G}$ or $\widetilde{\mathcal{G}}$ from $\triangleright$-priority relationships of the other (Section 4.2).

Although the results in this section do not depend in any way on the composite DAG framework of Theorem 2.1 (and the resulting scheduling algorithms in [16]), they imply that the theorem always applies to an arbitrary DAG $\mathcal{G}$ and its dual $\widetilde{\mathcal{G}}$ simultaneously (see Corollary 5.1 for a formal verification).

4.1 Scheduling-Based Duality

Let $\mathcal{G}$ be a DAG that has n nonsinks, comprising the set $U = \{u_1, u_2, \ldots, u_n\}$, and N nonsources, comprising the set $\{v_1, v_2, \ldots, v_N\}$. Let Σ be a schedule for $\mathcal{G}$ that executes $\mathcal{G}$'s nonsinks in the order

$$u_{k_1}, u_{k_2}, \ldots, u_{k_{n-1}}, u_{k_n}, \qquad (2)$$

after which, Σ executes all of $\mathcal{G}$'s sinks. Each node execution of a nonsink, say, u_{k_j}, renders ELIGIBLE a (possibly empty) "packet" of nonsources $P_j = \{v_{j,1}, \ldots, v_{j,i_j}\}$ of $\mathcal{G}$ (clearly, $\mathcal{G}$ must have an arc from u_{k_j} to each node in P_j). Thus, Σ renders $\mathcal{G}$'s nonsources ELIGIBLE in a sequence of such "packets":

$$\begin{aligned} &P_1 = \{v_{1,1}, \ldots, v_{1,i_1}\},\ P_2 = \{v_{2,1}, \ldots, v_{2,i_2}\},\ \ldots, P_n \\ &P_n = \{v_{n,1}, \ldots, v_{n,i_n}\}. \end{aligned} \tag{3}$$

A schedule $\widetilde{\Sigma}$ for $\mathcal{G}$'s dual DAG $\widetilde{\mathcal{G}}$ is *dual to* Σ if it executes $\widetilde{\mathcal{G}}$'s nonsinks—which, recall, are $\mathcal{G}$'s nonsources—in an order of the form

$$[[v_{n,1}, \ldots, v_{n,i_n}]], [[v_{n-1,1}, \ldots, v_{n-1,i_{n-1}}]], \ldots, [[v_{1,1}, \ldots, v_{1,i_1}]], \tag{4}$$

where $[[a, b, \ldots, c]]$ denotes a fixed but unspecified permutation of the set $\{a, b, \ldots, c\}$ (after which, $\widetilde{\Sigma}$ executes all of $\widetilde{\mathcal{G}}$'s sinks). Note that $\widetilde{\mathcal{G}}$ will generally admit many schedules that are dual to Σ. Note also that both Σ and $\widetilde{\Sigma}$ honor Lemma 2.1(a).

Theorem 4.1. *Let the DAG $\mathcal{G}$ admit the IC-optimal schedule $\Sigma_{\mathcal{G}}$. Any schedule for $\widetilde{\mathcal{G}}$ that is dual to $\Sigma_{\mathcal{G}}$ is IC optimal.*

Proof. Let $\Sigma_{\mathcal{G}}$ execute $\mathcal{G}$'s nonsinks in the order (2) and render $\mathcal{G}$'s nonsources ELIGIBLE according to the partial order specified implicitly by (3). It follows that every schedule for $\widetilde{\mathcal{G}}$ that is dual to $\Sigma_{\mathcal{G}}$ executes $\widetilde{\mathcal{G}}$'s N nonsinks in an order of the form (4).

Assume now for contradiction that some schedule for $\widetilde{\mathcal{G}}$, call it $\Sigma_{\widetilde{\mathcal{G}}}$, is dual to $\Sigma_{\mathcal{G}}$ but is *not* IC optimal. There must then be a schedule $\Sigma'_{\widetilde{\mathcal{G}}}$ for $\widetilde{\mathcal{G}}$ and a step $t \in [1, N-1]$ such that

$$E_{\widetilde{\mathcal{G}}'}(t) > E_{\widetilde{\mathcal{G}}}(t).$$

We show now that the existence of schedule $\Sigma'_{\widetilde{\mathcal{G}}}$ refutes the alleged IC optimality of schedule $\Sigma_{\mathcal{G}}$.

Let $s \in [1, n]$ be the smallest number of nonsinks of $\mathcal{G}$ that schedule $\Sigma_{\mathcal{G}}$ needs to execute in order to render $\geq N - t$ nonsources of $\mathcal{G}$ ELIGIBLE. In other words, $E_{\Sigma_{\mathcal{G}}}(s) \geq N - t$, but $E_{\Sigma_{\mathcal{G}}}(s') < N - t$ for all $s' < s$. We claim that

$$E_{\widetilde{\mathcal{G}}}(t) \geq n - s. \tag{5}$$

To verify this inequality, note that as $\Sigma_{\mathcal{G}}$ executes nonsinks u_{k_1}, u_{k_2}, ..., u_{k_s} of $\mathcal{G}$ during its first s steps, it renders ELIGIBLE the nonsources of $\mathcal{G}$ in the set $P_1 \cup P_2 \cup \cdots \cup P_s$ (see (2) and (3)). Since $E_{\Sigma_{\mathcal{G}}}(s) \geq N - t$, it follows that $t' \overset{\text{def}}{=} |P_{s+1} \cup P_{s+2} \cup \cdots \cup P_n| \leq t$.

Since $\Sigma_{\widetilde{\mathcal{G}}}$ is dual to $\Sigma_{\mathcal{G}}$, during its first t' steps, $\Sigma_{\widetilde{\mathcal{G}}}$ executes precisely the t' nonsinks of $\widetilde{\mathcal{G}}$ in the set $P_{s+1} \cup P_{s+2} \cup \cdots \cup P_n$. Because schedule $\Sigma_{\mathcal{G}}$ is IC optimal, each node in $\{u_{k_1}, u_{k_2}, \ldots, u_{k_s}\}$ must have in $\mathcal{G}$ an arc to one or more nodes in $P_1 \cup P_2 \cup \cdots \cup P_s$. Were this not the case, $\Sigma_{\mathcal{G}}$ could render $\geq N - t$ sinks of $\mathcal{G}$ ELIGIBLE by executing (at most) $s - 1$ nonsinks of $\mathcal{G}$. It thus follows that the set of nonsources of $\widetilde{\mathcal{G}}$ that are rendered ELIGIBLE by $\Sigma_{\widetilde{\mathcal{G}}}$'s first t' node executions is precisely the set $\{u_{k_{s+1}}, u_{k_{s+2}}, \ldots, u_{k_n}\}$ so that $E_{\Sigma_{\widetilde{\mathcal{G}}}}(t') = n - s$. Since $t \geq t'$, this verifies inequality (5).

Recall that, by hypothesis,

$$E_{\Sigma'_{\widetilde{\mathcal{G}}}}(t) > E_{\Sigma_{\widetilde{\mathcal{G}}}}(t).$$

There must then be a set $V = \{v_1, v_2, \ldots, v_t\}$ of t nonsinks of $\widetilde{\mathcal{G}}$ that schedule $\Sigma'_{\widetilde{\mathcal{G}}}$ executes during its first t steps, thereby rendering ELIGIBLE a set S comprising *at least* $n - s + 1$ nonsources of $\widetilde{\mathcal{G}}$. Clearly, there is no arc in $\widetilde{\mathcal{G}}$ from any node of $\overline{V}$ to any node of S. It follows that any schedule $\Sigma'_{\mathcal{G}}$ for $\mathcal{G}$ that executes the $s' \leq s - 1$ nonsinks of $\mathcal{G}$ in the set $\overline{S}$ during its first s' steps renders ELIGIBLE all of the nonsources of $\mathcal{G}$ in the set $\overline{V}$. However, these nonsources are at least $N - t$ in number. It follows that $E_{\Sigma'_{\mathcal{G}}}(s-1) > E_{\Sigma_{\mathcal{G}}}(s-1)$, which contradicts the alleged IC optimality of schedule $\Sigma_{\mathcal{G}}$.

We conclude that schedule $\Sigma_{\widetilde{\mathcal{G}}}$ cannot exist where the (arbitrary dual) schedule $\Sigma_{\widetilde{\mathcal{G}}}$ is IC optimal for $\widetilde{\mathcal{G}}$, as was claimed. □

The following corollary of Theorem 4.1 is immediate from the fact that $\widetilde{\widetilde{\mathcal{G}}} = \mathcal{G}$.

Corollary 4.1. *A DAG $\mathcal{G}$ admits an IC-optimal schedule if and only if its dual, $\widetilde{\mathcal{G}}$, does.*

4.2 Priority-Based Duality

We now derive $\triangleright$-priorities between two DAGs, $\mathcal{G}_1$ and $\mathcal{G}_2$, from $\triangleright$-priorities between their dual DAGs: $\widetilde{\mathcal{G}}_1$ and $\widetilde{\mathcal{G}}_2$, respectively. For this purpose, it is convenient to supplement the system of inequalities (1) that defines $\triangleright$-priority with a dual formulation of the relation.

Let $\mathcal{G}$ be a DAG that has N nonsources and let Σ be a schedule for $\mathcal{G}$. For any $e \in [0, N]$, let $X_\Sigma(e)$ be the smallest number of nonsinks of $\mathcal{G}$ that must be executed in order to render at least e nonsources of $\mathcal{G}$ ELIGIBLE. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be disjoint DAGs that respectively have N_1 and N_2 nonsources and admit IC-optimal schedules Σ_1 and Σ_2. If the following inequalities hold,

$$\begin{aligned} &(\forall e \in [0, N_1])\ (\forall f \in [0, N_2]) : \\ &\qquad X_{\Sigma_1}(e) + X_{\Sigma_2}(f) \ \geq\ X_{\Sigma_1}(\min\{e + f, N_1\}) \\ &\qquad\quad + X_{\Sigma_2}(\max\{0, e + f - N_1\}), \end{aligned} \tag{6}$$

then $\mathcal{G}_1$ *has dual priority over* $\mathcal{G}_2$, denoted $\mathcal{G}_1 \,\widetilde{\triangleright}\, \mathcal{G}_2$. Clearly, the relations $\widetilde{\triangleright}$ and $\triangleright$ are equivalent: the former relation strives to minimize the number of EXECUTED nonsinks for a given number of ELIGIBLE nonsources, and the latter strives to maximize the number of ELIGIBLE nonsources for a given number of EXECUTED nonsinks. Stated formally,

Lemma 4.1. *If the DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$ admit IC-optimal schedules Σ_1 and Σ_2, respectively, then $\mathcal{G}_1 \triangleright \mathcal{G}_2$ if and only if $\mathcal{G}_1 \,\widetilde{\triangleright}\, \mathcal{G}_2$.*

Lemma 4.1 affords us easy access to the following result (note the reversal of indices):

Theorem 4.2. *For all DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$, $\mathcal{G}_1 \triangleright \mathcal{G}_2$ if and only if $\widetilde{\mathcal{G}_2} \triangleright \widetilde{\mathcal{G}_1}$.*

Proof. For $i = 1, 2$, let $\mathcal{G}_i$ have n_i nonsinks and N_i nonsources and let it admit the IC-optimal schedule Σ_i. Theorem 4.1 tells us how we can construct from Σ_i an IC-optimal schedule $\widetilde{\Sigma}_i$ for $\widetilde{\mathcal{G}_i}$. Moreover, the proof of that theorem gives us valuable information about how Σ_i and $\widetilde{\Sigma}_i$ operate. Specifically (using the notation of the proof), recall that in order to render $N - j$ nonsources of $\widetilde{\mathcal{G}}$ ELIGIBLE, we must execute j nonsinks in packets $P_{j+1}, P_{j+2}, \ldots, P_N$. Equivalently, in order to render j nonsources of $\widetilde{\mathcal{G}}$ ELIGIBLE, we must execute j nonsinks in packets $P_{N-j+1}, P_{N-j+2}, \ldots, P_N$. Moreover, executing j nonsinks of $\mathcal{G}$ renders ELIGIBLE exactly the nonsources of packets $P_1, P_2, \ldots, P_j$. Hence, for all $e \in [0, n_i]$,

$$\begin{aligned} X_{\widetilde{\Sigma}_i}(e) &= |P_{n_i-e+1} \cup P_{n_i-e+2} \cup \cdots \cup P_{n_i}| \\ &= N_i - |P_1 \cup P_2 \cup \cdots \cup P_{n_i-e}| \\ &= N_i - E_{\Sigma_i}(n_i - e). \end{aligned}$$

Thus, for any $e \in [0, n_1]$ and $f \in [0, n_2]$, we have

$$X_{\widetilde{\Sigma}_i}(e) + X_{\widetilde{\Sigma}_2}(f) = (N_1 + N_2) - E_{\Sigma_1}(n_1 - e) - E_{\Sigma_2}(n_2 - f). \tag{7}$$

Say that $\mathcal{G}_1 \triangleright \mathcal{G}_2$. By system (1), we then have

$$\begin{aligned} &E_{\Sigma_1}(n_1 - e) + E_{\Sigma_2}(n_2 - f) \le \\ &\quad E_{\Sigma_1}(\min\{n_1, n_1 + n_2 - e - f\}) + E_{\Sigma_2}(\max\{0, n_2 - e - f\}). \end{aligned}$$

Combining this inequality with (7), we find that

$$\begin{aligned} X_{\widetilde{\Sigma}_1}&(e) + X_{\widetilde{\Sigma}_2}(f) \\ &\ge (N_1 + N_2) - E_{\Sigma_1}(\min\{n_1, n_1 + n_2 - e - f\}) \\ &\qquad - E_{\Sigma_2}(\max\{0, n_2 - e - f\}) \\ &= X_{\widetilde{\Sigma}_1}(n_1 - \min\{n_1, n_1 + n_2 - e - f\}) \\ &\qquad + X_{\widetilde{\Sigma}_2}(n_2 - \max\{0, n_2 - e - f\}) \\ &= X_{\widetilde{\Sigma}_1}(\max\{0, e + f - n_2\}) + \\ &\qquad + X_{\widetilde{\Sigma}_2}(\min\{n_2, e + f\}). \end{aligned}$$

This last inequality means that $\widetilde{\mathcal{G}_2} \mathrel{\widetilde{\triangleright}} \widetilde{\mathcal{G}_1}$ so that, by Lemma 4.1, $\widetilde{\mathcal{G}_2} \triangleright \widetilde{\mathcal{G}_1}$, as claimed. The converse follows "by running the argument backward." □

5 Where We Are and Where We Are Going

5.1 Conclusions

The results in this paper significantly expand, via two avenues, the range of DAGs that the algorithmic theory in [16] can schedule IC optimally.

Expanding the repertoire of building blocks. In addition to the structurally uniform DAGs in Fig. 2 and the simple composite DAGs in [16], we are now able to schedule IC optimally any sequence of *sums of PBTs* that is linearly ordered under $\triangleright$-priority $\mathcal{P}_1 \triangleright \mathcal{P}_2 \triangleright \cdots \triangleright \mathcal{P}_n$. As but one significant illustration, we are now able to schedule IC optimally three classes of DAGs, which are reminiscent of the DAGs encountered in a variety of scientific computations. Let us be given

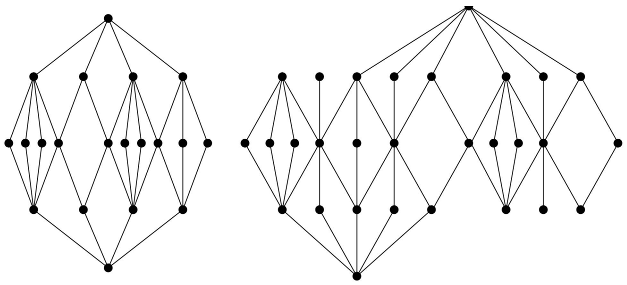


Fig. 6. Two DAGs that can now be scheduled IC optimally. Edges represent arcs that point upward.

- any sequence of *sums of W-strands* that is linearly ordered under $\triangleright$-priority $\breve{\mathcal{S}}_1 \triangleright \breve{\mathcal{S}}_2 \triangleright \cdots \triangleright \breve{\mathcal{S}}_n$ and
- any sequence of *sums of M-strands* that is linearly ordered under $\triangleright$-priority $\widetilde{\mathcal{S}_1} \triangleright \widetilde{\mathcal{S}_2} \triangleright \cdots \triangleright \widetilde{\mathcal{S}_m}$.

Then, any composite DAG of one of the three types admits the IC-optimal schedule dictated by Theorem 2.1:

1. $\breve{\mathcal{S}}_1 \Uparrow \cdots \Uparrow \breve{\mathcal{S}}_n$,
2. $\breve{\mathcal{S}}_1 \Uparrow \cdots \Uparrow \breve{\mathcal{S}}_m$, and
3. $\breve{\mathcal{S}}_1 \Uparrow \cdots \Uparrow \breve{\mathcal{S}}_n \Uparrow \breve{\mathcal{S}}_1 \Uparrow \cdots \Uparrow \breve{\mathcal{S}}_n$.

Informally, we now have a scheduling control over a broad family of DAGs that are expansive, reductive, and expansive-reductive (in the way that, for example, many series-parallel [17] or fork-join DAGs are). Fig. 6 exhibits two simple DAGs (constructed from those in Fig. 4) that exemplify the expanded repertoire of DAGs that we can schedule IC optimally because of our expanded repertoire of building blocks. In the figure, the left-hand DAG is composite of type $\mathcal{W}[4] \Uparrow \mathcal{W}[4,2,4,3] \Uparrow \mathcal{M}[4,2,4,3] \Uparrow \mathcal{M}[4]$, and the right-hand DAG is composite of type

$$\begin{aligned} \mathcal{W}[5] \Uparrow \widetilde{\mathcal{P}}&\left[\frac{1}{4}, 3, \frac{1}{3}, 3, \frac{1}{2}, 2, \frac{1}{4}, 3, \frac{1}{2}\right] \\ &\Uparrow \mathcal{P}\left[\frac{1}{4}, 3, \frac{1}{3}, 3, \frac{1}{2}, 2, \frac{1}{4}, 3, \frac{1}{2}\right] \Uparrow \mathcal{M}[6]. \end{aligned}$$

Straightforward calculations using (1) show that $\mathcal{W}[4] \triangleright \mathcal{W}[4,2,4,3] \triangleright \mathcal{M}[4,2,4,3] \triangleright \mathcal{M}[4]$ and that

$$\begin{aligned} \mathcal{W}[5] \triangleright \widetilde{\mathcal{P}}&\left[\frac{1}{4}, 3, \frac{1}{3}, 3, \frac{1}{2}, 2, \frac{1}{4}, 3, \frac{1}{2}\right] \\ &\triangleright \mathcal{P}\left[\frac{1}{4}, 3, \frac{1}{3}, 3, \frac{1}{2}, 2, \frac{1}{4}, 3, \frac{1}{2}\right] \triangleright \mathcal{M}[6]. \end{aligned}$$

Exploiting duality to schedule DAGs optimally. We have shown how we can "read off" an IC-optimal schedule for an arbitrary DAG $\mathcal{G}$ from an IC-optimal schedule for $\mathcal{G}$'s dual. We have also shown how we can "read off" $\triangleright$-priorities among a collection of DAGs from $\triangleright$-priorities among their dual DAGs. This allows one to shortcut the algorithmic suite from [16] and its upcoming sequel [4] when dealing with DAGs whose duals have previously been dealt with. Alternatively, one can actually incorporate the duality into the algorithmic suite, as suggested by the following corollary to Theorems 4.1 and 4.2.

Corollary 5.1. *If the DAG $\mathcal{G}$ admits an IC-optimal schedule via Theorem 2.1, then so does its dual DAG $\widetilde{\mathcal{G}}$.*

Proof. The premise of the corollary implies that the DAG $\mathcal{G}$ is a composite of type $\mathcal{G}_1 \Uparrow \mathcal{G}_2 \Uparrow \cdots \Uparrow \mathcal{G}_n$ for some bipartite DAGs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$ that admit IC-optimal schedules $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, respectively and that form a linear chain of $\triangleright$-priorities $\mathcal{G}_1 \triangleright \mathcal{G}_2 \triangleright \cdots \triangleright \mathcal{G}_n$. Consider the bipartite DAGs $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2, \ldots, \widetilde{\mathcal{G}}_n$ that are respectively dual to the DAGs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$.

We note first that the DAG $\widetilde{\mathcal{G}}$ that is dual to $\mathcal{G}$ is composite of type $\widetilde{\mathcal{G}}_n \Uparrow \widetilde{\mathcal{G}}_{n-1} \Uparrow \cdots \Uparrow \widetilde{\mathcal{G}}_1$. We leave the verification of this easy fact to the reader.

Next, by Theorem 4.1, we know that for each $i \in [1, n]$, the DAG $\widetilde{\mathcal{G}}_i$ admits an IC-optimal schedule: specifically, one that is dual to the IC-optimal schedule Σ_i of $\mathcal{G}_i$.

Finally, by Theorem 4.2, the DAGs $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2, \ldots, \widetilde{\mathcal{G}}_n$ form a linear chain of $\triangleright$-priorities $\widetilde{\mathcal{G}}_n \triangleright \widetilde{\mathcal{G}}_{n-1} \triangleright \cdots \triangleright \widetilde{\mathcal{G}}_1$.

In summary, $\mathcal{G}$'s dual DAG $\widetilde{\mathcal{G}}$ satisfies the conditions of Theorem 2.1 whenever $\mathcal{G}$ does and hence admits an IC-optimal schedule via the formula of that theorem. □

5.2 Projections

Our work on this project proceeds in several directions.

Theory. We are engaged in investigations aimed at extending the scope of the theory in [16] in a number of ways:

1. We are expanding the scheduling component of our theory to move further beyond its current dependence on $\triangleright$-linear compositions of building blocks. Our new work focuses on (IC-optimal) schedules that allow the *interleaved* execution of building blocks. Preliminary results appear in [4].
2. We are seeking a rigorous framework for devising schedules that are "approximately" IC optimal. This thrust is important for computational reasons (a computationally simple heuristic may be "almost as good" as a more arduously derived IC-optimal schedule) and because many DAGs do not admit IC-optimal schedules.
3. We are working to extend our theory so that it can optimally schedule composite DAGs whose building blocks are not necessarily bipartite.

Simulations and experiments. We are engaged in a suite of simulation experiments that seek to determine the extent to which our scheduling algorithms actually enhance the efficiency of Internet-based computations. A report on simulations involving real scientific DAGs appears in [14], and a report on artificially generated ones appears in [10].

Integration with grid schedulers. Our most ambitious nontheoretical endeavor involves incorporating our suite of scheduling algorithms into a real scheduling tool: the Condor DAGMan tool [3]. Coauthor Malewicz developed a tool [14] for prioritizing the jobs of a DAGMan file while visiting Argonne National Laboratory.

Acknowledgments

A portion of the research of G. Cordasco and G. Malewicz was done while visiting the University of Massachusetts Amherst. A portion of the research of G. Malewicz was done while visiting the Mathematics and Computer Science Division of Argonne National Laboratory. The research of G. Malewicz was supported in part by the US National Science Foundation (NSF) Grant ITR-800864. The research of A. Rosenberg was supported in part by the NSF Grant CCF-0342417. The authors are grateful to Ian Foster and Michael Wilde (Argonne National Laboratory) and to Matt Yurkewych (National Security Agency) for valuable conversations at several stages of the research reported here.

References

[1] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service-Oriented Grid Computing," *Proc. 10th Heterogeneous Computing Workshop (HCW '01),* 2001.
[2] W. Cirne and K. Marzullo, "The Computational Co-Op: Gathering Clusters into a Metacomputer," *Proc. 13th Int'l Parallel Processing Symp. (IPPS '99),* pp. 160-166, 1999.
[3] Condor Project, Univ. of Wisconsin, http://www.cs.wisc.edu/condor, 2007.
[4] G. Cordasco, G. Malewicz, and A.L. Rosenberg, "Extending IC-Scheduling Theory via the Sweep Algorithm," Univ. of Massachusetts, 2007, submitted for publication.
[5] G. Cordasco, G. Malewicz, and A.L. Rosenberg, "Applying IC-Scheduling Theory to Some Familiar Computations," *Proc. Workshop Large-Scale, Volatile Desktop Grids (PCGrid '07),* 2007.
[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. MIT Press, 1999.
[7] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure,* second ed. Morgan Kaufmann, 2004.
[8] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. Supercomputer Applications,* 2001.
[9] L. Gao and G. Malewicz, "Toward Maximizing the Quality of Results of Dependent Tasks Computed Unreliably," *Theory of Computing Systems,* to appear, 2007.
[10] R. Hall, A.L. Rosenberg, and A. Venkataramani, "A Comparison of Dag-Scheduling Strategies for Internet-Based Computing," *Proc. 22nd Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2007.
[11] H.T. Hsu, "An Algorithm for Finding a Minimal Equivalent Graph of a Digraph," *J. ACM,* vol. 22, pp. 11-16, 1975.
[12] D. Kondo, H. Casanova, E. Wing, and F. Berman, "Models and Scheduling Mechanisms for Global Computing Applications," *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS '02),* 2002.
[13] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home: Massively Distributed Computing for SETI," P.F. Dubois, ed., *Computing in Science and Eng.,* IEEE CS Press, 2000.
[14] G. Malewicz, I. Foster, A.L. Rosenberg, and M. Wilde, "A Tool for Prioritizing DAGMan Jobs and Its Evaluation," *J. Grid Computing,* vol 5., no. 2, pp. 197-212, 2007.
[15] G. Malewicz and A.L. Rosenberg, "On Batch-Scheduling Dags for Internet-Based Computing," *Proc. 11th Euro-Par Conf. (Euro-Par '05),* pp. 262-271, 2005.
[16] G. Malewicz, A.L. Rosenberg, and M. Yurkewych, "Toward a Theory for Scheduling Dags in Internet-Based Computing," *IEEE Trans. Computing,* vol. 55, pp. 757-768, 2006.
[17] M. Mitchell, "Creating Minimal Vertex Series Parallel Graphs from Directed Acyclic Graphs," *Proc. Australasian Symp. Information Visualisation (InVis.au '04),* pp. 133-139, 2004.
[18] A.L. Rosenberg, "On Scheduling Mesh-Structured Computations for Internet-Based Computing," *IEEE Trans. Computing,* vol. 53, pp. 1176-1186, 2004.
[19] A.L. Rosenberg and M. Yurkewych, "Guidelines for Scheduling Some Common Computation-Dags for Internet-Based Computing," *IEEE Trans. Computing,* vol. 54, pp. 428-438, 2005.

[20] X.-H. Sun and M. Wu, "GHS: A Performance Prediction and Node Scheduling System for Grid Computing," *Proc. 17th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '03)*, 2003.

Gennaro Cordasco received the *Laurea* degree (*cum laude*) in 2002 and the Dottorato di Ricerca (PhD degree) in 2006, both in informatica (computer science) from the University of Salerno. In 2005, he visited the Department of Computer Science at the University of Massachusetts, Amherst, doing research activity under the supervision of distinguished Professor Arnold L. Rosenberg. His research interests focus on distributed algorithms, parallel data structures, peer-to-peer systems, and Internet-based computing.

Grzegorz Malewicz received the BA degrees in computer science and in applied mathematics in 1996 and 1998, respectively, and the MS degree in computer science in 1998, all from the University of Warsaw. He received the PhD degree in computer science from the University of Connecticut in 2003. He is an engineer at Google. He has had internships at the AT&T Shannon Laboratory (summer 2001) and Microsoft Corp. (summer 2000 and fall 2001). He visited the Laboratory for Computer Science, Massachusetts Institute of Technology (MIT, academic year 2002–2003), and was a visiting scientist at the University of Massachusetts, Amherst (summer 2004) and Argonne National Laboratory (summer 2005). He was an assistant professor at the University of Alabama, where he taught computer science from 2003 until 2005. His research focuses on high-performance parallel and distributed computing, experimental and theoretical algorithmics, combinatorial optimization, and scheduling. His research appears in top journals and conferences and includes a singly authored *SIAM Journal on Computing* paper that solves a decade-old problem in distributed computing. He is a member of the IEEE.

Arnold L. Rosenberg received the AB degree in mathematics from Harvard College in 1962 and the AM and PhD degrees in applied mathematics from Harvard University in 1963 and 1966, respectively. He is a distinguished university professor of computer science at the University of Massachusetts (UMass), 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 Watson Research Center from 1965 to 1981. He has held visiting positions at Yale University and the University of Toronto, was a Lady Davis Visiting Professor at the Technion (Israel Institute of Technology) in 1994, and was a Fulbright Research Scholar at the University of Paris-South in 2000. He has served as coeditor of several books. His research focuses on developing algorithmic models and techniques to deal with the new modalities of "collaborative computing" (wherein multiple computers cooperate to solve a computational problem) that result from emerging technologies, especially Internet-based computing. He is the author or coauthor of more than 150 technical papers on these and other topics in theoretical computer science and discrete mathematics. He 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.