Extending IC-Scheduling via the Sweep Algorithm*

Gennaro Cordasco†
Università di Salerno
Dip. di Informatica ed Applicazioni
Fisciano, SA 84084, ITALY
cordasco@dia.unisa.it

Grzegorz Malewicz
Google Inc.
Dept. of Engineeering
Mountain View, CA 94043, USA
malewicz@google.com

Arnold L. Rosenberg
Colorado State University
Dept. of Electrical and Computer Engineering
Fort Collins, CO 80523, USA
rsnbrg@colostate.edu

*A portion of this paper was presented at *PDP 2008* [10].

†Correspondence to: Gennaro Cordasco, Dip. di Informatica ed Applicazioni, Università di Salerno, 84084 Fisciano, Italy. E-mail: cordasco@dia.unisa.it Tel: +39-089-969307 Fax: +39-089-969600.

Abstract

A key challenge when scheduling computations over the Internet is *temporal unpredictability:* remote "workers" arrive and depart at unpredictable times and often provide unpredictable computational resources; the time for communication over the Internet is impossible to predict accurately. In response, earlier research has developed the underpinnings of a theory of how to schedule computations having intertask dependencies in a way that renders tasks eligible for execution at the maximum possible rate. Simulation studies suggest that such scheduling: (*a*) utilizes resource providers' computational resources well, by enhancing the likelihood of having work to allocate to an available client; (*b*) lessens the likelihood of a computation's stalling for lack of tasks that are eligible for execution. The applicability of the current version of the theory is limited by its demands on the structure of the DAG that models the computation being scheduled—namely, that the DAG be decomposable into *connected* bipartite "building-block" DAGs. The current paper extends the theory by developing the *Sweep Algorithm*, which takes a significant step toward removing this restriction. The resulting augmented suite of scheduling algorithms allows one to craft optimal schedules for a large range of DAGs that the earlier framework could not handle. Most of the newly optimally scheduled DAGs presented here are artificial but "close" in structure to DAGs that arise in real computations; one of the new DAGs is a component of a large DAG that arises in a functional Magnetic Resonance Imaging application.

Keywords. IC-scheduling, IC-scheduling Theory, Internet-based computing, Grid computing, Global computing, Scheduling DAGs, Theory.

1 Introduction

Earlier work [28, 29] has developed the concept of *IC-scheduling*, a formal framework for studying the problem of scheduling computations that have intertask dependencies, for the several modalities of *Internet-based computing* (*IC*, for short). These modalities include Volunteer computing [23], Peer-to-Peer computing (P2P) [32], Global computing [7] and Grid computing [6, 12, 13]. Acknowledging the *temporal unpredictability* of IC—communication is over the Internet, and resource providers are usually not dedicated to the computation being scheduled—IC-scheduling strives to craft schedules that 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 resource providers' computational resources, by always having work to allocate to an available client; (b) lessening the likelihood of a computation's stalling pending completion of already-allocated tasks. Two simulation studies—[24], which focuses on scheduling a small number of real scientific computations, and [17], which derives eligibility-enhancing schedules for hundreds of artificially generated computations—suggest that schedules produced under IC-scheduling often have marked computational benefits over schedules produced by a variety of common heuristics (including FIFO scheduling).

Inspired by the case studies of [28, 29], the study in [26] turned IC-scheduling from a collection of ad hoc optimal schedules for specific computation-DAGs to an algorithmic framework that provides optimal schedules for a broad class of DAGs for IC. The development in [26] begins with any collection of *building-block* DAGs that we know how to schedule optimally. It develops two algorithmic notions that allow us to schedule complex computation-DAGs built from these building blocks. (Section 2.2 provides technical details.)

1. *The priority relation $\triangleright$ on pairs of* DAG*s.* 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* DAG*s.* If one uses composition to construct

a complex computation-DAG $\mathcal{G}$ from a set of building blocks that are pairwise comparable under the relation $\triangleright$, then we can often compute an optimal schedule for $\mathcal{G}$ from optimal schedules for the building blocks.

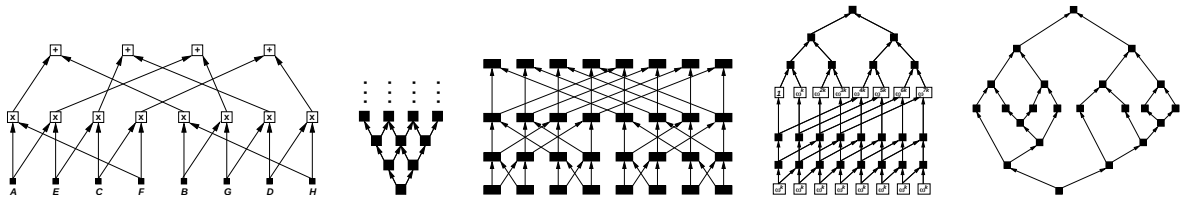

Figure 1: *Data-dependency* DAG*s for five familiar computations:* (left to right) *recursive matrix multiplication, a wavefront computation, the Fast-Fourier Transform (FFT), the discrete Laplace transform, a divide-and-conquer computation.*

The development of IC-scheduling is a work in progress, aimed at providing theoretical underpinnings that will ultimately enable the design of systems that can speed up the performance of IC platforms. In its current state, IC-scheduling already produces optimal schedules for DAGs that arise within a large variety of disparate, important computations; Fig. 1 depicts five such DAGs whose optimal schedules are derived in [9, 26, 28, 29]. Fig. 2 presents three artificial DAGs that are also scheduled optimally using the algorithmic framework of [8, 26]; these DAGs illustrate that IC-scheduling does not demand the high degree of structural uniformity observed in the DAGs of Fig. 1. The results of the simulation-based studies of [24, 17] suggest that there would be significant practical benefits to using a scheduler based on IC-scheduling to schedule important computations that are reused frequently. Moreover, implementing the simulations in the studies has suggested that using such a scheduler would not be computationally onerous.

The successes of the current version of IC-scheduling notwithstanding, there remain many significant computation-DAGs that admit optimal schedules (via ad hoc analyses) that the current framework cannot develop optimal schedules for. Fig. 3 presents three such DAGs. The top two DAGs in the figure are artificial ones that we consider here because of their superficial structural similarities to DAGs such as those in Figs. 1 and 2. The bottom DAG

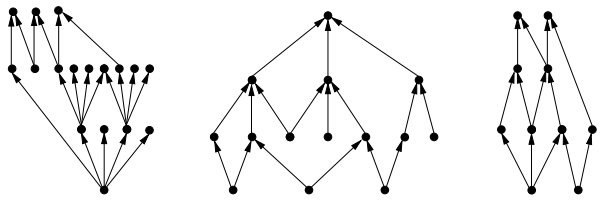

Figure 2: *Three composite* DAG*s that the framework of [8, 26] can schedule optimally.*

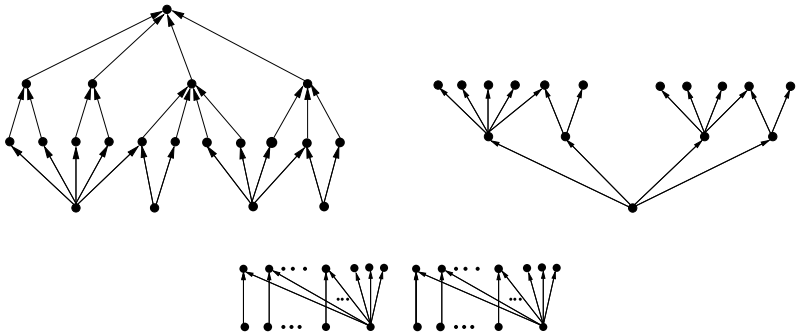

Figure 3: *Three* DAG*s that the framework of [8, 26] cannot schedule optimally, while our extended framework can:* (top) *two artificial* DAG*s;* (bottom) *a disconnected* DAG *inspired by a functional MRI application.*

is a replicated version of a component of a large functional Magnetic Resonance Imaging computation-DAG; see [24] for a more detailed description of this DAG. (While only a single instance of this last DAG appears in the fMRI computation-DAG, similar DAGs appear multiple times, so the replicated version of this DAG is consistent with the spirit of the complete computation-DAG.)

> *Clarification:* It is easy to produce *locally* optimal schedules for DAGs such as these. What was lacking prior to the current study were *globally* optimal schedules.

The reason that the current version of IC-scheduling cannot produce *optimal* schedules for DAGs such as those in Fig. 3 is that, as we demonstrate in Section 5, every optimal schedule for these DAGs must *interleave* the execution of each DAG's building blocks—and the current version of IC-scheduling is unable to produce schedules that perform such interleaving.

The current paper adds to the scheduling framework of [8, 26] an algorithmic tool (see Section 3) that significantly expands the repertoire of DAGs that IC-scheduling can schedule optimally. This new tool results from algorithms that allow one: (a) to craft schedules that *interleave* the execution of subDAGs that have no interdependencies, (b) to schedule using bipartite building-block DAGs that *need not be connected*. In particular, the new framework can schedule the three DAGs of Fig. 3 optimally. We remark that the new tool *does not change the order of complexity of the IC-scheduling framework*. The tool is executed once, to generate a large number of (possibly disconnected) building blocks that can be used for every DAG decomposition; the tool is, thus, particularly important for computations (such as linear-algebraic ones) that will be performed many times. Moreove, as we show in Section 5.2, the Sweep Algorithm can sometimes even *speed up* the IC-scheduling process.

The Sweep Algorithm is just one new "brick" in the edifice of IC-scheduling, but it provides a significant new scheduling tool. The development of the algorithm and its applications demands a rather sophisticated analysis, to which this paper is devoted. By expanding the class of DAGs that can be scheduled optimally via IC-scheduling, the Algorithm allows us to capture a larger repertoire of computation-DAGs, including some computationally very important ones that appear in practical applications. As suggested by Fig. 3, the new version of IC-scheduling can optimally schedule a much-broadened class of divide-and-conquer DAGs; and the fMRI-inspired DAGs of the figure are representative of a very significant class of computations that only now can be scheduled optimally.

Related work. The problem of scheduling a parallel program on a parallel/distributed computing system, with the goal of minimizing job completion time, has been studied since the advent of such systems; see, e.g., [19, 30] for many approaches to the problem. IC-scheduling shares its overall strategy with the myriad approaches that schedule a complex computation by decomposing it into subcomputations and developing a schedule for the big computation by "composing" schedules for the smaller ones. Countless heuristics have been developed for these usually **NP**-Hard problems, and many studies have attempted to

analyze and compare such heuristics [15, 16, 20, 27]; a taxonomy of scheduling problems and approaches appears in [21]. Despite the disparities in approach to scheduling described in the cited sources, virtually every algorithm/heuristic that predates IC-scheduling shared one central characteristic: they all relied on knowing (almost) exact times for each computer to execute each subcomputation and to communicate its results with a collaborating computer. The central premise underlying IC-scheduling is that within the world of IC computing, one cannot even approach such exact knowledge. This premise is shared by sources such as [22, 23], whose approaches to scheduling for IC platforms admit margins of error in time estimates of 50% or more. Indeed, IC-scheduling is an analytically based proposal for what to do when accurate estimates are out of the question.

Most closely related to the current study are its companions in developing IC-scheduling. The topic is introduced in [28, 29], and optimal schedules are characterized for expansive and reductive trees and meshes and for the FFT DAG. We have already described the role of [26] in beginning to develop IC-scheduling as an algorithmic framework. This framework is extended significantly in [8], both by allowing one to exploit duality[1] as a scheduling tool and by greatly expanding the repertoire of building blocks accessible to the framework of [26]. Motivated by the fact that many DAGs do not admit an optimal schedule in the sense of IC-scheduling, the study in [25] proposes a *batch*-oriented scheduling regimen for IC. Several sources focus on specific challenges encountered when scheduling DAGs for IC: one finds in [14] a probabilistic approach to the problem of executing tasks on unreliable clients; a framework for minimizing makespan when processors proceed asynchronously on DAGs with unit-time tasks is studied and illustrated in [3]. Novel approaches to scheduling computations having no intertask dependencies appear in many sources, including [1, 2, 4, 5]. Finally, the impetus for our study derives from the many exciting systems- and/or application-oriented studies of IC, in sources such as [6, 7, 12, 13, 22, 23, 31].

[1] The *dual* of a DAG $\mathcal{G}$ is obtained by reversing all of $\mathcal{G}$'s arcs.

2 A Basis for a Scheduling Theory

2.1 A Scheduling Model for IC

Computation-DAGs. A[2] DAG $\mathcal{G}$ has a set $N_{\mathcal{G}}$ of *nodes*, each representing a task in a computation, and a set $A_{\mathcal{G}}$ of *arcs*, each representing an intertask dependency. For arc $(u \rightarrow v) \in A_{\mathcal{G}}$:

- task v cannot be executed until task u is;
- u is a *parent* of v, and v is a *child* of u in $\mathcal{G}$.

The *indegree* (resp., *outdegree*) of $u \in N_{\mathcal{G}}$ is its number of parents (resp., children). A parentless node is a *source*; a childless node is a *sink*. $\mathcal{G}$ is *bipartite* if $N_{\mathcal{G}}$ can be partitioned into X and Y, and each arc $(u \rightarrow v)$ has $u \in X$ and $v \in Y$. $\mathcal{G}$ is *connected* if it is so when one ignores arc orientations. When $N_{\mathcal{G}_1} \cap N_{\mathcal{G}_2} = \emptyset$, the *sum* $\mathcal{G}_1 + \mathcal{G}_2$ of DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$ is the DAG with node-set $N_{\mathcal{G}_1} \cup N_{\mathcal{G}_2}$ and arc-set $A_{\mathcal{G}_1} \cup A_{\mathcal{G}_2}$.

Schedules and their quality. When one executes a DAG $\mathcal{G}$, a node $v \in N_{\mathcal{G}}$ becomes ELIGIBLE (for execution) as soon as all of its parents have been executed. (Hence, sources are always ELIGIBLE.) We do not allow recomputation of nodes, so a node loses its ELIGIBILITY once it is executed. In compensation, after node $v \in N_{\mathcal{G}}$ has been executed, new nodes may be rendered ELIGIBLE; this occurs when v is their last parent to be executed. Informally, a *schedule* for $\mathcal{G}$ is a rule for selecting which ELIGIBLE node to execute at each step of an execution of $\mathcal{G}$; formally, it is a *topological sort* of $\mathcal{G}$, i.e., a linearization of $N_{\mathcal{G}}$ under which all arcs point from left to right. (*We measure time in an event-driven manner*, as the number of nodes that have been executed thus far.) Letting $E_\Sigma(t)$ denote the number of ELIGIBLE nonsources on $\mathcal{G}$ after the tth node-execution under Σ, we define the *profile* associated with schedule Σ for $\mathcal{G}$ as follows. For $t \in [1, |N_{\mathcal{G}}|]$,[3]

$$E_\Sigma = (E_\Sigma(1), E_\Sigma(2), \ldots, E_\Sigma(|N_{\mathcal{G}}|))$$

[2] For brevity, we henceforth refer to "DAGs," without the qualifier "computation."

[3] $[a, b]$ denotes the set of integers $\{a, a+1, \ldots, b\}$.

We measure Σ's quality at step t by the size of $E_\Sigma(t)$: the larger, the better. Our goal is to execute $\mathcal{G}$'s nodes in an order that maximizes quality *at every step* $t \in [1, |N_\mathcal{G}|]$ *of the execution*. Informally, a schedule Σ that achieves this demanding goal is **IC-optimal**; formally, Σ is **IC-optimal** if for each schedule Σ' for $\mathcal{G}$, $E_\Sigma(t) \geq E_{\Sigma'}(t)$ for every $t \in [1, |N_\mathcal{G}|]$. Note that distinct IC-optimal schedules, Σ_1 and Σ_2, for $\mathcal{G}$ have *identical associated profiles:* $E_{\Sigma_1} = E_{\Sigma_2}$.

The significance of IC optimality stems from two scenarios. (1) Schedules that produce ELIGIBLE nodes more quickly may reduce the chance of a computation's stalling because resource providers are slow—so no new tasks can be allocated pending the return of already allocated ones. (2) If the IC Server receives several requests for tasks at (roughly) the same time, then having more ELIGIBLE tasks available allows the Server to satisfy more requests, thereby increasing "parallelism." The simulations in [24, 17] bolster our hope that this intuition does indeed enhance the speed of IC computations.

2.2 A Framework for Crafting IC-Optimal Schedules

Simplifying the search for schedules. We lose no IC quality by executing all sources of a bipartite DAG before any sinks.

Lemma 2.1 ([26])**.** *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 the IC quality of the resulting profile is no less than* Σ*'s.*

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

$$\begin{aligned}&(\forall x \in [0, s_1])\ (\forall y \in [0, s_2]) : \\ &\quad E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \ \leq\ E_{\Sigma_1}(\min\{s_1, x+y\}) + E_{\Sigma_2}(\max\{0, x+y-s_1\}),\end{aligned} \tag{2.1}$$

then $\mathcal{G}_1$ *has priority over* $\mathcal{G}_2$, denoted $\mathcal{G}_1 \triangleright \mathcal{G}_2$.[4] Informally, one never decreases IC quality by executing a source of $\mathcal{G}_1$ whenever possible. Importantly: *relation* $\triangleright$ *is transitive* [26].

[4] By definition, one can decide in time proportional to $s_1 s_2$ whether or not $\mathcal{G}_1 \triangleright \mathcal{G}_2$.

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

- Start with a set $\mathcal{B}$ of base DAGs.

  (The base DAGs considered in [8, 26] are *connected bipartite* DAGs. We term each such base DAG a *CBBB*, for "Connected Bipartite Building Block.")
- 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* (or, disjoint union), $\mathcal{G}_1 + \mathcal{G}_2$, of the DAGs $\mathcal{G}_1, \mathcal{G}_2$. Rename nodes to ensure that $N_{\mathcal{G}}$ is disjoint from $N_{\mathcal{G}_1}$ and $N_{\mathcal{G}_2}$.
  - Select some set S_1 of sinks from the copy of $\mathcal{G}_1$ in the sum $\mathcal{G}_1 + \mathcal{G}_2$, and an equal-size set S_2 of sources from the copy of $\mathcal{G}_2$ in the sum.
  - Pairwise identify (i.e., 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; the induced set of arcs is $\mathcal{G}$'s arc-set.[5]
- 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* $[\mathcal{G}_1 \Uparrow \mathcal{G}_2]$. Clearly, *the composition operation is associative;* i.e., $\mathcal{G}$ is composite of type $[[\mathcal{G}_1 \Uparrow \mathcal{G}_2] \Uparrow \mathcal{G}_3]$ if, and only if, it is composite of type $[\mathcal{G}_1 \Uparrow [\mathcal{G}_2 \Uparrow \mathcal{G}_3]]$.

The DAG $\mathcal{G}$ is a $\triangleright$-*linear composition* of the connected bipartite DAGs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ if: (a) $\mathcal{G}$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$; (b) $\mathcal{G}_i \triangleright \mathcal{G}_{i+1}$, for all $i \in [1, n-1]$.

Theorem 2.1 ([26])**.** *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 nonsinks of $\mathcal{G}_i$, in the order mandated by Σ_i.*
2. *Σ finally executes all sinks of $\mathcal{G}$ in any order.*

[5] An arc $(u \to v)$ is *induced* if $\{u, v\} \subseteq N_{\mathcal{G}}$.

One finds in [26] 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. In summary, the algorithms operate as follows on a given DAG $\mathcal{G}$.

1. The first algorithm, borrowed from [18], "prunes" $\mathcal{G}$ to remove every arc $a = (u, v)$ that is a *shortcut*, in the sense that there is a path from u to v that does not use arc a. It is shown in [26] that the resulting "skeleton" DAG $\mathcal{G}'$ shares all of its IC-optimal schedules (if any exist) with $\mathcal{G}$.
2. The second algorithm "parses" $\mathcal{G}'$ into CBBBs $\mathcal{G}_1, \ldots, \mathcal{G}_n$, such that $\mathcal{G}'$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$, or it determines that no such "parsing" is possible.
3. The third algorithm replaces $\mathcal{G}'$ by its *super*-DAG $\mathcal{G}''$. The nodes of $\mathcal{G}''$ are the CBBBs $\mathcal{G}_1, \ldots, \mathcal{G}_n$; its arcs form a blueprint of the sequence of compositions that created $\mathcal{G}'$. Specifically, if $\mathcal{G}'$ was formed by identifying some of the sinks of $\mathcal{G}_i$ with some of the sources of $\mathcal{G}_j$, then there is an arc from supernode $\mathcal{G}_i$ to supernode $\mathcal{G}_j$ in $\mathcal{G}''$.
4. The final algorithm determines whether or not there is an $\triangleright$-linearization of the CBBBs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ that is consistent with the topological dependencies within $\mathcal{G}''$. This means that if some $\mathcal{G}_i$ precedes some $\mathcal{G}_j$ in a topological sort of $\mathcal{G}''$, then $\mathcal{G}_i \triangleright \mathcal{G}_j$ in the $\triangleright$-linearization.

The early success of [26] in scheduling significant DAGs (including those in Figs. 1 and 2) leads to the current challenge of expanding the range of DAGs that we can schedule IC optimally, especially DAGs that occur in practical applications.

3 The IC-Sweep Algorithm

3.1 Overview

Throughout this section, focus on a *sequence*[6] of $p \geq 2$ disjoint DAGs, $\mathcal{G}_1, \ldots, \mathcal{G}_p$, that have, respectively, $n_1, \ldots, n_p$ nonsinks, and that all admit IC-optimal schedules. This section develops Algorithm IC-Sweep, an algorithm that efficiently accomplishes several tasks that play an essential role in advancing the DAG-scheduling framework of Section 2.2.

IC-optimal scheduling. Algorithm IC-Sweep can enable one either to craft an IC-optimal schedule for a sum-dag $\mathcal{G} = \mathcal{G}_1 + \cdots + \mathcal{G}_p$ or to prove that no such schedule exists. Notably, the algorithm thereby gives one access to IC-optimal schedules that cannot be derived from the framework provided by Theorem 2.1. One can now develop IC-optimal schedules for many DAGs whose building blocks must be executed in an *interleaved* fashion in any IC-optimal schedule. Indeed, the DAGs' building blocks need no longer always be CBBBs: they can now also be *sums* of CBBBs. The earlier framework required CBBB building blocks that were executed without interleaving.

Speeding up certain scheduling procedures. Algorithm IC-Sweep can significantly accelerate certain procedures that allow one to schedule in a divide-and-conquer manner. An example of this benefit appears in Section 5.2.

Deciding $\triangleright$-priorities. Algorithm IC-Sweep provides a nicely structured, efficient mechanism for determining whether or not the DAGs $\mathcal{G}_1, \ldots, \mathcal{G}_p$ form a $\triangleright$-*priority chain*; i.e., do we have $\mathcal{G}_1 \triangleright \cdots \triangleright \mathcal{G}_p$ or not?

Timing. Our references to Algorithm IC-Sweep's efficiency mean specifically that, when given IC-optimal schedules $\Sigma_1, \ldots, \Sigma_p$ for $\mathcal{G}_1, \ldots, \mathcal{G}_p$ (respectively), and their associated

[6]The order in which the DAGs are presented *is significant*.

profiles, $E_{\Sigma_1}, \ldots, E_{\Sigma_p}$, the algorithm operates on the sum $\mathcal{G}$ within time[7]

$$O\left(\sum_{1 \le i < j \le p} n_i n_j\right) \quad = \quad O\left((n_1 + \cdots + n_p)^2\right). \tag{3.1}$$

3.2 Algorithm IC-Sweep

We begin to develop Algorithm IC-Sweep by simplifying the search space within which the algorithm seeks an IC-optimal schedule for the sum $\mathcal{G} = \mathcal{G}_1 + \cdots + \mathcal{G}_p$.

Lemma 3.1. *If the sum $\mathcal{G} = \mathcal{G}_1 + \cdots + \mathcal{G}_p$ admits an IC-optimal schedule Σ, then, for each $i \in [1, p]$, Σ must execute the nonsinks of $\mathcal{G}$ that come from $\mathcal{G}_i$ in the same order as some IC-optimal schedule Σ_i for $\mathcal{G}_i$.*

Proof. We simplify exposition while conveying all relevant ideas by focusing on the case $p = 2$. Let Σ be an arbitrary IC-optimal schedule for $\mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2$ that honors Lemma 2.1, by executing all nonsinks before any sinks. For each $t \in [1, n_1 + n_2]$, by step t, Σ will have executed some $k \le t$ nonsinks that come from $\mathcal{G}_1$ and $t - k$ nonsinks that come from $\mathcal{G}_2$. The total number of ELIGIBLE nodes at step t will be the sum of the numbers of nodes of $\mathcal{G}_1$ and $\mathcal{G}_2$ that are rendered ELIGIBLE by executing those nonsinks. This sum can, by definition, not decrease if the k nonsinks from $\mathcal{G}_1$ are those that some IC-optimal schedule Σ_1 for $\mathcal{G}_1$ would have executed, and the $t - k$ nonsinks from $\mathcal{G}_2$ are those that some IC-optimal schedule Σ_2 for $\mathcal{G}_2$ would have executed. □

Lemma 3.1 indicates that the secret to finding IC-optimal schedules for sums of DAGs that individually admit IC-optimal schedules is to discover how to *interleave* the executions of the nodes of the individual DAGs.

Focus henceforth on the IC-optimal schedules $\Sigma_1, \ldots, \Sigma_p$ for $\mathcal{G}_1, \ldots, \mathcal{G}_p$, respectively

[7] All uses of asymptotic notation (big O, big Ω) allow all specified variables to grow without bound.

3.2.1 Algorithm IC-Sweep on two DAGs

To simplify exposition, we first describe Algorithm 2-IC-Sweep, the two-DAG version of Algorithm IC-Sweep (i.e., the case $p = 2$.)

Algorithm 2-IC-Sweep

1. Use the profiles associated with schedules Σ_1 and Σ_2 to construct an $(n_1 + 1) \times (n_2 + 1)$ table $\mathcal{E}$ such that:

   $(\forall i \in [0, n_1])(\forall j \in [0, n_2])$

   $\mathcal{E}(i, j)$ is the maximum number of nodes of $\mathcal{G}_1 + \mathcal{G}_2$ that can be rendered ELIGIBLE by the execution of i nonsinks of $\mathcal{G}_1$ and j nonsinks of $\mathcal{G}_2$.

   By Lemma 3.1, we can construct $\mathcal{E}$ as follows:

   $$(\forall i \in [0, n_1])(\forall j \in [0, n_2]) \quad \mathcal{E}(i, j) \;=\; E_{\Sigma_1}(i) + E_{\Sigma_2}(j).$$

   /*An initial portion of $\mathcal{E}$ appears in Table 1. (Recall that $E_\Sigma(0) \equiv 0$.)*/

2. Perform a left-to-right pass along each diagonal $i + j$ of $\mathcal{E}$ in turn, and fill in the $(n_1 + 1) \times (n_2 + 1)$ *Verification Table* $\mathcal{V}$, as follows.

   (a) Initialize all $\mathcal{V}(i, j)$ to "NO"

   (b) Set $\mathcal{V}(0, 0)$ to "YES"

   (c) **for each** $t \in [1, n_1 + n_2]$:

      i. **for each** $\mathcal{V}(i, j)$ with $i + j = t$:

         if $\mathcal{V}(i - 1, j) =$ "YES" **or** $\mathcal{V}(i, j - 1) =$ "YES"

         and if $\mathcal{E}(i, j) \;=\; \max_{a+b=t}\{\mathcal{E}(a, b)\}$

         then set $\mathcal{V}(i, j)$ to "YES"

         /*A rectilinear continuation has been found*/

      ii. **if** no entry $\mathcal{V}(i, j)$ with $i + j = t$ has been set to "YES"

         then HALT and report "There is no IC-optimal schedule."

         /*A diagonal of "NO" entries precludes a rectilinear path*/

(d) HALT and report "There is an IC-optimal schedule."

$\mathcal{E}$	0	1	$\cdots$	n_2
0	$E_{\Sigma_1}(0) + E_{\Sigma_2}(0)$	$E_{\Sigma_1}(0) + E_{\Sigma_2}(1)$	$\cdots$	$E_{\Sigma_1}(0) + E_{\Sigma_2}(n_2)$
1	$E_{\Sigma_1}(1) + E_{\Sigma_2}(0)$	$E_{\Sigma_1}(1) + E_{\Sigma_2}(1)$	$\cdots$	$E_{\Sigma_1}(1) + E_{\Sigma_2}(n_2)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n_1	$E_{\Sigma_1}(n_1) + E_{\Sigma_2}(0)$	$E_{\Sigma_1}(n_1) + E_{\Sigma_2}(1)$	$\cdots$	$E_{\Sigma_1}(n_1) + E_{\Sigma_2}(n_2)$

Table 1: *An initial portion of the Table $\mathcal{E}$ constructed by Algorithm* 2-IC-Sweep.

We claim that Algorithm 2-IC-Sweep achieves the claimed goals: the desired decision procedure for the existence of an IC-optimal schedule for $\mathcal{G}_1 + \mathcal{G}_2$ and for the existence of a $\triangleright$-priority relation between the DAGs (in either direction).

Theorem 3.1. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be disjoint* DAG*s that admit, respectively, the IC-optimal schedules Σ_1 and Σ_2 and that have, respectively, n_1 and n_2 nonsinks. Algorithm* 2-IC-Sweep *determines, within time $O(n_1 n_2)$, whether or not the sum $\mathcal{G}_1 + \mathcal{G}_2$ admits an IC-optimal schedule. In the positive case, the Algorithm provides such a schedule.*

Proof. By Lemma 3.1, as we determine whether or not $\mathcal{G}_1 + \mathcal{G}_2$ admits an IC-optimal schedule, we may restrict attention to schedules that execute the nonsinks of $\mathcal{G}_1$ in the same order as does Σ_1 and the nonsinks of $\mathcal{G}_2$ in the same order as does Σ_2. Algorithm 2-IC-Sweep does precisely this, by using optimal profiles as it constructs Table $\mathcal{E}$.

The requirement that an IC-optimal schedule Σ produce maximally many ELIGIBLE nodes at every step $t \in [1, n_1 + n_2]$ is, easily, equivalent to the assertion that Σ's sequence of node-executions specify a special type of *rectilinear* path within Table $\mathcal{E}$, viz., a sequence of Table entries obtained via a sequence of downward or rightward moves. This special path

must connect entry $\mathcal{E}(0,0)$ and entry $\mathcal{E}(n_1,n_2)$ while having each entry $\mathcal{E}(i,j)$ satisfy

$$\mathcal{E}(i,j) \;=\; \max_{a+b=i+j}\{\mathcal{E}(a,b)\}.$$

Algorithm 2-IC-Sweep uses the Verification Table $\mathcal{V}$ to search for just such a path. Indeed, any rectilinear path of "YES" entries in successive diagonals of Table $\mathcal{V}$ specifies an IC-optimal schedule for $\mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2$. Specifically: a downward move mandates executing the next nonsink of $\mathcal{G}_1$ that is mandated by Σ_1; a rightward move mandates executing the next nonsink of $\mathcal{G}_2$ that is mandated by Σ_2. Moreover, the absence of such a rectilinear path indicates that no schedule for $\mathcal{G}$ maximizes the number of ELIGIBLE nodes at every step.

The timing of Algorithm 2-IC-Sweep is obvious because a constant-time computation is performed at every entry of $\mathcal{V}$. We leave the detailed analysis to the reader. □

By Theorem 2.1, one way for $\mathcal{G}_1 + \mathcal{G}_2$ to admit an IC-optimal schedule is if either $\mathcal{G}_1 \rhd \mathcal{G}_2$, or $\mathcal{G}_2 \rhd \mathcal{G}_1$, or both. Algorithm 2-IC-Sweep can be used to detect these special cases:

- $\mathcal{G}_1 \rhd \mathcal{G}_2$ if, and only if, Table $\mathcal{V}$ contains a path of "YES" entries from position $\mathcal{V}(0,0)$ to position $\mathcal{V}(n_1,n_2)$, that is shaped like an uppercase "L," i.e., that performs n_1 downward moves followed by n_2 rightward moves.
- $\mathcal{G}_2 \rhd \mathcal{G}_1$ if, and only if, Table $\mathcal{V}$ contains a path of "YES" entries from position $\mathcal{V}(0,0)$ to position $\mathcal{V}(n_1,n_2)$, that is shaped like the digit "7," i.e., that performs n_2 rightward moves followed by n_1 downward moves.

Note. *Because all IC-optimal schedules for a* DAG *produce the same profiles, Algorithm* 2-IC-Sweep *produces the same tables, hence reaches the same conclusions in the same amount of time, no matter which IC-optimal schedules one uses for the summand* DAGs $\mathcal{G}_1$ *and* $\mathcal{G}_2$.

3.2.2 Algorithm IC-Sweep on multiple DAGs

A naive extension of Algorithm 2-IC-Sweep that operates on a sum of $p > 2$ DAGs, $\mathcal{G} = \mathcal{G}_1 + \cdots + \mathcal{G}_p$, would extend the 2-dimensional tables $\mathcal{E}$ and $\mathcal{V}$ to p-dimensional tables. A

thus-extended algorithm would operate in time $\Omega(n_1 \times \cdots \times n_p)$. We gain efficiency by rewriting $\mathcal{G}$ in the form $(\cdots((\mathcal{G}_1 + \mathcal{G}_2) + \mathcal{G}_3) + \cdots + \mathcal{G}_p)$ and using Algorithm 2-IC-Sweep iteratively to "sweep-analyze" the DAGs according to this expression. Thus implemented, the algorithm operates in the (generally) much-lower time (3.1). This strategy applies Algorithm 2-IC-Sweep to the components of $\mathcal{G}$ seriatim, via the following regimen.

Algorithm IC-Sweep

1. Perform Algorithm 2-IC-Sweep on the sum $\mathcal{G}_1 + \mathcal{G}_2$.
2. For each step $k = 2, 3, \ldots$:
   (a) **if** Step $k-1$ does not succeed—i.e., the $(k-1)$th invocation of Algorithm 2-IC-Sweep halts with the answer "NO"—**then** HALT and give the answer "NO"
   (b) **else if** Step $k-1$ succeeds—i.e., the $(k-1)$th invocation of Algorithm 2-IC-Sweep halts with the answer "YES"—**then** perform Algorithm 2-IC-Sweep on the sum $(\mathcal{G}_1 + \cdots + \mathcal{G}_k) + \mathcal{G}_{k+1}$.

Theorem 3.2. *Let $\mathcal{G}_1, \ldots, \mathcal{G}_p$ be $p \geq 2$ disjoint* DAG*s, that admit, respectively, the IC-optimal schedules $\Sigma_1, \ldots, \Sigma_p$ and that have, respectively, $n_1, \ldots, n_p$ nonsinks. Algorithm* IC-Sweep *determines, within time bounded above by (3.1):*

1. *whether or not the sum $\mathcal{G}_1 + \cdots + \mathcal{G}_p$ admits an IC-optimal schedule; in the positive case, the Algorithm provides an optimal schedule;*
2. *whether or not $\mathcal{G}_1 \rhd \cdots \rhd \mathcal{G}_p$.*

Proof. We simplify expressions via the following abbreviations: $\mathcal{G}_{1,1} = \mathcal{G}_1$, and, inductively, $\mathcal{G}_{1,k} = \mathcal{G}_{1,k-1} + \mathcal{G}_k$. Algorithm IC-Sweep processes, in turn, $\mathcal{G}_1 + \mathcal{G}_2$, then $\mathcal{G}_{1,2} + \mathcal{G}_3$, then $\mathcal{G}_{1,3} + \mathcal{G}_4$, and so on, until it has finally processed $\mathcal{G} \stackrel{\text{def}}{=} \mathcal{G}_{1,p-1} + \mathcal{G}_p = \mathcal{G}_1 + \cdots + \mathcal{G}_p$.

To prove the theorem, let us imagine that we have the p-dimensional analogue, $\mathcal{E}_p$, of the 2-dimensional table $\mathcal{E}$ of Algorithm 2-IC-Sweep. $\mathcal{E}_p$ is obtained from the optimal profile of

each DAG $\mathcal{G}_j$; in detail, using *any* IC-optimal schedule Σ_j for each $\mathcal{G}_j$:

$$\mathcal{E}_p(i_1, \ldots, i_p) \;=\; E_{\Sigma_1}(i_1) + E_{\Sigma_2}(i_2) + \cdots + E_{\Sigma_p}(i_p). \tag{3.2}$$

1. IC-optimality. As in the proof of Theorem 3.1, one shows easily that the sum $\mathcal{G}$ admits an IC-optimal schedule if, and only if, there is an $(n_1 + \cdots + n_p)$-entry *rectilinear* path in Table $\mathcal{E}_p$ that connects entries $\mathcal{E}_p(0, \ldots, 0)$ and $\mathcal{E}_p(n_1, \ldots, n_p)$, whose jth element, for each $j \in [0,\ n_1 + \cdots + n_p]$, has maximum value within the jth "diagonal hyperplane," $\{\langle i_1, \ldots, i_p\rangle \mid i_1 + \cdots + i_p \;=\; j\}$. In other words, there is a sequence of table entries each of whose indices increments precisely one coordinate, and whose jth element is maximum over all table entries $\mathcal{E}_p(i_1, \ldots, i_p)$ with $i_1 + \cdots + i_p = j$.

We verify by induction on p that Algorithm IC-Sweep correctly determines whether or not $\mathcal{G}$ admits an IC-optimal schedule. By Theorem 3.1, Algorithm 2-IC-Sweep makes the correct 2-DAG determination at each step of Algorithm IC-Sweep. This yields the base case, $p = 2$, so we need verify only that $\mathcal{G}$ admits an IC-optimal schedule if, and only if, for each $k \in [2, p-1]$, each "prefix" sum $\mathcal{G}_{1,k}$ admits an IC-optimal schedule.

We extend the induction by establishing a correspondence between the p-dimensional table $\mathcal{E}_p$ and the 2-dimensional table $\mathcal{E}$ created when Algorithm IC-Sweep invokes Algorithm 2-IC-Sweep on the sum $\mathcal{G}_{1,p-1} + \mathcal{G}_p$. The generic entry, $\mathcal{E}_p(i_1, \ldots, i_p)$, of $\mathcal{E}_p$ is given in (3.2); assuming, by induction, that $\mathcal{G}_{1,p-1}$ admits an IC-optimal schedule, $\Sigma_{1,p-1}$ and that

$$\begin{aligned}
\mathcal{E}((i_1 + \ldots + i_{p-2}), i_{p-1}) \;&=\; E_{\Sigma_{1,p-1}}(i_1 + \ldots + i_{p-1}) \\
&=\; \max_{j_1+\ldots+j_{p-2}=i_1+\ldots+i_{p-2}} \mathcal{E}_{p-1}(j_1, \ldots, j_{p-2}, i_{p-1}),
\end{aligned}$$

the generic entry, $\mathcal{E}((i_1 + \ldots + i_{p-1}), i_p)$, of $\mathcal{E}$ is given by

$$\mathcal{E}((i_1 + \ldots + i_{p-1}), i_p) \;=\; \mathcal{E}((i_1 + \ldots + i_{p-2}), i_{p-1}) + E_{\Sigma_p}(i_p).$$

Since our inductive hypothesis tells us that

$$\begin{aligned}
\mathcal{E}((i_1 + \ldots + i_{p-2}), i_{p-1}) \;&=\; E_{\Sigma_{1,p-1}}(i_1 + \cdots + i_{p-1}) \\
&=\; \max_{j_1+\ldots+j_{p-2}=i_1+\ldots+i_{p-2}} \mathcal{E}_{p-1}(j_1, \ldots, j_{p-2}, i_{p-1}),
\end{aligned}$$

we see that

$$\begin{aligned}\mathcal{E}((i_1 + \ldots + i_{p-1}), i_p) &= E_{\Sigma_{1,p-1}}(i_1 + \cdots + i_{p-1}) + E_{\Sigma_p}(i_p) \\ &= E_{\Sigma_{1,p}}(i_1 + \cdots + i_p) \\ &= \max_{j_1+\ldots+j_{p-1}=i_1+\ldots+i_{p-1}} \mathcal{E}_p(j_1, \ldots, j_{p-1}, i_p).\end{aligned}$$

This extends our induction: Algorithm IC-Sweep makes correct decisions about the existence of IC-optimal schedules. Moreover, it produces such a schedule if one exists. To wit: after each step $k \in [2, p-1]$, the algorithm provides an IC-optimal schedule for $\mathcal{G}_{1,k}$, if one exists; hence, after step p, we have an IC-optimal schedule for $\mathcal{G}_{1,p} = \mathcal{G}$, if one exists.

2. $\triangleright$-priorities. The $\triangleright$-priority chain, $\mathcal{G}_1 \triangleright \cdots \triangleright \mathcal{G}_p$, exists if, and only if, the Verification table $\mathcal{V}_p$, constructed in the natural way from $\mathcal{E}_p$, contains a dimension-by-dimension path of "YES" entries, from $\mathcal{V}_p(0, 0, \ldots, 0)$ to $\mathcal{V}_p(n_1, \ldots, n_{k+1})$, that covers, in turn,

$$\begin{array}{rcccc}
\text{all dimension-1 entries:} & \mathcal{V}_p(0,0,\ldots,0), & \ldots, & \mathcal{V}_p(n_1,0,\ldots,0), \\
\text{then all dimension-2 entries:} & \mathcal{V}_p(n_1,0,\ldots,0), & \ldots, & \mathcal{V}_p(n_1,n_2,\ldots,0), \\
\vdots & \vdots & \vdots & \vdots \\
\text{then all dimension-}p\text{ entries:} & \mathcal{V}_p(n_1,n_2,\ldots,0), & \ldots, & \mathcal{V}_p(n_1,n_2,\ldots,n_p).
\end{array}$$

Instead of explicitly generating the p-dimensional table $\mathcal{V}_p$ and searching for a dimension-by-dimension path of "YES" entries, Algorithm IC-Sweep constructs a sequence of two-dimensional tables $\mathcal{V}_2^{(2)}, \ldots, \mathcal{V}_2^{(p)}$ and, at each stage $k \in [2, p]$, checks if $\mathcal{V}_2^{(k)}$ contains a path of "YES" entries from $\mathcal{V}_2^{(k)}(0, 0)$ to $\mathcal{V}_2^{(k)}(n_1 + \cdots + n_k,\ n_{k+1})$, that is shaped like an uppercase "L." The presence of the kth such path indicates that $\mathcal{G}_{i,k} \triangleright \mathcal{G}_{k+1}$. We leave to the reader the induction on k that invokes the transitivity of $\triangleright$ to establish the following.

Claim. *For each $k \in [2, p]$, if $\mathcal{G}_{1,k} \triangleright \mathcal{G}_{k+1}$, then $\mathcal{G}_1 \triangleright \cdots \triangleright \mathcal{G}_{k+1}$.*

3. Timing. Our first invocation of Algorithm 2-IC-Sweep sweep-analyzes $\mathcal{G}_1$ and $\mathcal{G}_2$, hence takes time $T_{1,2} \leq \alpha n_1 n_2$, for some constant $\alpha > 0$ that is characteristic of the algorithm and independent of the sizes of the swept DAGs . Our second invocation sweep-analyzes $(\mathcal{G}_1 + \mathcal{G}_2)$ and $\mathcal{G}_3$, hence takes time $T_{1,3} \leq \alpha(n_1 + n_2)n_3$. Inductively, our kth

invocation of the algorithm sweep-analyzes $(\mathcal{G}_1 + \cdots + \mathcal{G}_{k-2})$ and $\mathcal{G}_{k-1}$, hence takes time $T_{1,k} \leq \alpha(n_1 + \cdots + n_{k-2})n_{k-1}$. Collectively, then, Algorithm IC-Sweep operates within time

$$\sum_{i=2}^{p} T_{1,i} \;\leq\; \alpha \sum_{1 \leq i < j \leq p} n_i n_j,$$

thus verifying the claimed time (3.1). *Note that this timing analysis is independent of the order in which the* DAG*s* $\mathcal{G}_i$ *are sequenced for the sweep.* □

3.3 Sample Applications of Algorithm IC-Sweep

3.3.1 A sum of DAGs that requires interleaved scheduling

Consider two CBBBs: $\mathcal{B}_1$, which appears in Fig. 4(a), and $\mathcal{B}_2$, which appears in Fig. 4(b). The DAGs are "swept" in Table 2. One easily discerns the following three facts from the

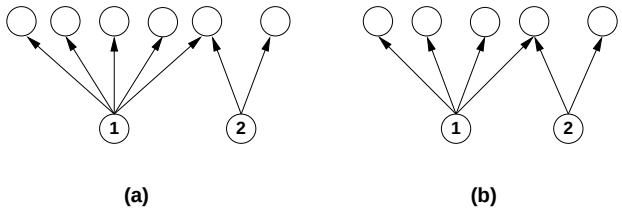

Figure 4: *Two CBBBs whose sum requires an interleaved schedule:* (a) $\mathcal{B}_1$; (b) $\mathcal{B}_2$.

$\mathcal{B}_1$ $\mathcal{B}_2 \rightarrow$ $\downarrow$	0	1	2
0	**0**	3	5
1	**4**	**7**	**9**
2	6	**9**	**11**

Table 2: *The Table* $\mathcal{E}$ *constructed by Algorithm* 2-IC-Sweep *for the CBBBs* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *of Fig. 4. The boxed entries indicate an IC-optimal schedule for* $\mathcal{B}_1 + \mathcal{B}_2$.

rectilinear paths between entries $\mathcal{E}(0,0)$ and $\mathcal{E}(2,2)$ in Table 2.

1. *Neither* $\mathcal{B}_1 \rhd \mathcal{B}_2$ *nor* $\mathcal{B}_2 \rhd \mathcal{B}_1$.

   No path shaped like an "L" or a "7" describes an IC-optimal schedule.

2. *The sum* $\mathcal{B}_1 + \mathcal{B}_2$ *admits an IC-optimal schedule.*

   The highlighted path contains only diagonal-maximizing entries.

3. *Both IC-optimal schedules for* $\mathcal{B}_1 + \mathcal{B}_2$ *(corresponding to the two rectilinear paths in Table* $\mathcal{E}$*) execute node* 1 *of* $\mathcal{B}_1$ *in the first step and node* 1 *of* $\mathcal{B}_2$ *in the second step.*

We thus see that the sum $\mathcal{B}_1 + \mathcal{B}_2$ admits an IC-optimal schedule—but only an interleaved one. This verifies that Algorithm IC-Sweep gives us algorithmic access to IC-optimal schedules that IC-scheduling, as presented in Section 2.2 cannot produce. In Section 4, we incorporate Algorithm IC-Sweep into IC-scheduling, thereby affording one access to interleaved IC-optimal schedules.

3.3.2 DAGs that admit no IC-optimal schedule

One can use Algorithm 2-IC-Sweep to show that the DAG $\mathcal{G}_3$ of Fig. 5(a) does not admit an

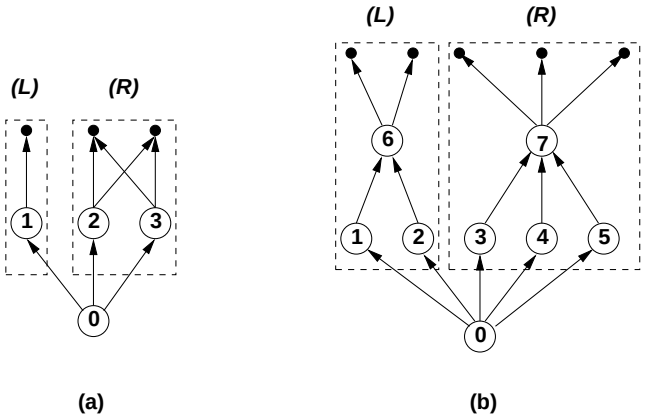

Figure 5: *Two* DAGs *that cannot be scheduled IC optimally:* (a) $\mathcal{G}_3$; (b) $\mathcal{G}_4$.

IC-optimal schedule. To wit, every execution of $\mathcal{G}_3$ must begin by executing node 0, $\mathcal{G}_3$'s only source. Having executed node 0, one is left with the challenge of optimally executing

the sum-DAG $\mathcal{G}_3^{(L)} + \mathcal{G}_3^{(R)}$, where $\mathcal{G}_3^{(L)}$ (resp., $\mathcal{G}_3^{(R)}$) is the sub-DAG of $\mathcal{G}_3$ within the lefthand (resp., the righthand) dashed box in Fig. 5(a). One sees in Table 3 that $\mathcal{G}_3^{(L)} + \mathcal{G}_3^{(R)}$ does not admit an IC-optimal schedule. Informally, in order to maximize the number of ELIGIBLE

$\mathcal{G}_3^{(L)}$ $\mathcal{G}_3^{(R)} \rightarrow$ $\downarrow$	0	1	2
0	**0**	0	**2**
1	**1**	1	**3**

Table 3: *The Table $\mathcal{E}$ constructed by Algorithm* 2-IC-Sweep *for the left and right sub-*DAG*s of $\mathcal{G}_3$; cf. Fig. 5(a). Boxed entries indicate maxima along diagonals.*

nonsources at step 1 of an execution of $\mathcal{G}_3^{(L)} + \mathcal{G}_3^{(R)}$, node 1 (in the figure) must be the first-executed node. However, in order to maximize the analogous number at step 2, nodes 2 and 3 (in the figure) must be the first two executed nodes.

A more complicated application of Algorithm 2-IC-Sweep, which employs the same reasoning as we just employed with $\mathcal{G}_3$, shows that the DAG $\mathcal{G}_4$ of Fig. 5(b) does not admit an IC-optimal schedule. Assume, for contradiction that it did admit the IC-optimal schedule Σ. Easily, Σ would execute node 0, $\mathcal{G}_4$'s unique source, first. From that point on, Σ would have to craft an IC-optimal schedule for the sum-DAG $\mathcal{G}_4^{(L)} + \mathcal{G}_4^{(R)}$.[8] Now, every IC-optimal schedule for $\mathcal{G}_4^{(L)}$ executes nodes $1, 2$ (in either order) and then executes node 6; similarly, every IC-optimal schedule for $\mathcal{G}_4^{(R)}$ executes nodes $3, 4, 5$ (in some order) and then executes node 7. We can, therefore, use Algorithm 2-IC-Sweep to determine whether or not $\mathcal{G}_4^{(L)} + \mathcal{G}_4^{(R)}$ admits an IC-optimal schedule—and we do so in Table 4. Easily, the Table shows that the sum does *not* admit an IC-optimal schedule.

[8] $\mathcal{G}_4^{(L)}$ (resp., $\mathcal{G}_4^{(R)}$) is the sub-DAG of $\mathcal{G}_4$ within the lefthand (resp., righthand) dashed box in Fig. 5(b).

$\mathcal{G}_4^{(L)}$ $\mathcal{G}_4^{(R)} \rightarrow$ $\downarrow$	0	1	2	3	4
0	[0]	[0]	0	1	[3]
1	[0]	0	0	1	[3]
2	[1]	1	1	2	[4]
3	[2]	2	2	3	[5]

Table 4: *The Table $\mathcal{E}$ constructed by Algorithm* 2-IC-Sweep *for the left and right boxed sub*DAG*s of $\mathcal{G}_4$; cf. Fig. 5(b). Boxed entries indicate maxima along diagonals.*

4 Scheduling DAGs Using Algorithm IC-Sweep

Algorithm IC-Sweep advances IC-scheduling theory by taking a major step toward liberating the theory from its dependence on *connected* building blocks (the only type appearing in [8, 26]). One can often now employ interleaved schedules for *sums* of CBBBs. This section is devoted to developing this extended capability.

4.1 The Enabling Theorem

One can often infer $\triangleright$-priorities between sums of DAGs, even if one does not have such priorities among the DAGs within each sum.

Theorem 4.1. *Let us be given $p+1$* DAG*s, $\mathcal{G}_1, \ldots, \mathcal{G}_p$, and $\mathcal{G}'$ such that:*

- *for each $i \in [1,p]$, $\mathcal{G}_i$ has n_i nonsinks and admits the IC-optimal schedule Σ_i; let $n = n_1 + \cdots + n_p$;*
- *$\mathcal{G}'$ has n' nonsinks and admits the IC-optimal schedule Σ';*
- *the sum $\mathcal{G} \stackrel{def}{=} \mathcal{G}_1 + \cdots + \mathcal{G}_p$ admits the IC-optimal schedule Σ.*

If $\mathcal{G}_i \triangleright \mathcal{G}'$ for all $i \in [1,p]$, then $\mathcal{G} \triangleright \mathcal{G}'$.

Proof. By Lemma 3.1, we may assume that Σ honors the order of nonsink-executions of $\Sigma_1, \ldots, \Sigma_p$. Consider a regimen for executing the nonsinks of both $\mathcal{G}$ and $\mathcal{G}'$, using Σ for the former and Σ' for the latter. Focus on a step t when

- For each $i \in [1,p]$, Σ has executed x_i nonsinks from $\mathcal{G}_i$; let $x \stackrel{\text{def}}{=} x_1 + \cdots + x_p$.
- Σ' has executed y nonsinks from $\mathcal{G}'$.

Clearly, $t = x + y$. The posited $\rhd$-priorities among the $\mathcal{G}_i$'s and $\mathcal{G}'$ allow us to derive the following chain of (in)equalities, which holds whenever $y > 0$ and $x_i < n_i$ for all $i \in [1,p]$.

$$\begin{aligned} E_\Sigma(x) + E_{\Sigma'}(y) &= E_\Sigma(x_1 + \cdots + x_p) \;+\; E_{\Sigma'}(y) \\ &= \Big(E_{\Sigma_1}(x_1) + \cdots + E_{\Sigma_p}(x_p)\Big) \;+\; E_{\Sigma'}(y) \\ &\le E_\Sigma\left(x + \sum_{i=1}^{p} (\min\{n_i, x_i + y\} - x_i)\right) + E_{\Sigma'}\left(\sum_{i=1}^{p} \max\{0, x_i + y - n_i\}\right). \end{aligned}$$

We iterate the transfer of nonsink-executions from $\mathcal{G}'$ to $\mathcal{G}$ for as long as possible, i.e., as long as $y > 0$ and, for some $i \in [1,p]$, $x_i < n_i$. Since $\mathcal{G}_i \rhd \mathcal{G}'$ for all $i \in [1,p]$, once this process has terminated, we shall either have reduced y to 0, having thereby increased x to $x + y$, or we shall have increased x to $n_1 + \cdots + n_p$, at which point no further transfers are possible. Thereby, we shall have transformed the sum $E_\Sigma(x) + E_{\Sigma'}(y)$ to the sum $E_\Sigma(\min\{n, x+y\}) + E_{\Sigma'}(\max\{0, x+y-n'\})$. Because the described process never decreases the then-current value of the sum $E_\Sigma(x) + E_{\Sigma'}(y)$, we infer that $\mathcal{G} \rhd \mathcal{G}'$, as claimed. □

An induction that repeatedly invokes the transitivity of $\rhd$ yields the following, which ensures that our expanded scheduling algorithm subsumes the algorithm of [26].

Corollary 4.1. *Say that each of the $p + q$ DAGs, $\mathcal{G}_1, \ldots, \mathcal{G}_p$ and $\mathcal{G}'_1, \ldots, \mathcal{G}'_q$, admits an IC-optimal schedule.*

If $\quad \mathcal{G}_1 \rhd \cdots \rhd \mathcal{G}_p \rhd \mathcal{G}'_1 \rhd \cdots \rhd \mathcal{G}'_q$

then $\quad (\mathcal{G}_1 + \cdots + \mathcal{G}_p) \rhd (\mathcal{G}'_1 + \cdots + \mathcal{G}'_q)$.

4.2 The Scheduling Consequences of Theorem 4.1

In the light of Theorem 4.1, Algorithm IC-Sweep allows us to expand the algorithmic suite of Section 2.2 and, thereby, to greatly expand the range of DAGs that we know how to schedule IC optimally. Our new scheduling regimen proceeds as follows.

1. Invoke the first three algorithms of Section 2.2 to produce, in succession:
   (a) the "pruned," shortcut-free DAG $\mathcal{G}'$,
   (b) the constituent CBBBs $\mathcal{B}_1, \ldots, \mathcal{B}_n$ of $\mathcal{G}'$, each admitting an IC-optimal schedule (whenever possible),
   (c) the super-DAG $\mathcal{G}''$ whose nodes are $\mathcal{B}_1, \ldots, \mathcal{B}_n$ and whose arcs indicate compositions thereof.
2. If the super-DAG $\mathcal{G}''$ cannot be generated—because some constituent subalgorithm or condition fails—then the new strategy does not produce a schedule for $\mathcal{G}$.
3. We seek to decompose $\mathcal{G}''$ into CBBBs, or sums thereof, that witness a $\triangleright$-linearization of $\mathcal{G}''$. We start with:
   - $\mathcal{G}''$, as the *current remnant super*-DAG $\mathcal{R}$,
   - an empty list L, as our current progress toward a $\triangleright$-linearization of $\mathcal{G}''$.

   Let $\mathcal{R}_{\mathcal{B}}$ denote the $\mathcal{R}$ obtained by *removing* source-CBBB $\mathcal{B}$ from the current $\mathcal{R}$.
   (a) If some source-CBBB $\mathcal{B}_i$ of $\mathcal{R}$ satisfies: for each source-CBBB $\mathcal{B}_j$ of $\mathcal{R}_{\mathcal{B}_i}$, $\mathcal{B}_i \triangleright \mathcal{B}_j$, then delete $\mathcal{B}_i$ from $\mathcal{R}$ (i.e., $\mathcal{R} \leftarrow \mathcal{R}_{\mathcal{B}_i}$) and append it to list L. Go to step (a).
   (b) If $\mathcal{R}$ is empty, then we are done: L is the desired $\triangleright$-linearization of $\mathcal{G}''$.
   (c) Say that we reach a point where, for each source-CBBB $\mathcal{B}_i$ of $\mathcal{R}$, there is a source-CBBB $\mathcal{B}_j$ of $\mathcal{R}_{\mathcal{B}_i}$ such that $\mathcal{B}_i \not\triangleright \mathcal{B}_j$. Then we attempt to extend L via Theorem 4.1—since we cannot just append a new source-CBBB. To this end:
      i. Assemble all source-CBBBs, $\mathcal{B}'_1, \ldots, \mathcal{B}'_k$ of $\mathcal{R}$.

ii. Say that (A) $\mathcal{B}' = \mathcal{B}'_1 + \cdots + \mathcal{B}'_k$ admits an IC-optimal schedule; (B) $\mathcal{B}'_i \rhd \mathcal{B}_j$ for all $i \in [1, k]$, and for each source-CBBB $\mathcal{B}_j$ of $\mathcal{R}_{\mathcal{B}'}$.[9] Then we invoke Theorem 4.1 to infer that $\mathcal{B}' \rhd \mathcal{B}_j$ for each source-CBBB $\mathcal{B}_j$ of $\mathcal{R}_{\mathcal{B}'}$; we can, therefore append $\mathcal{B}'$ to list L. We then return to step (a).

iii. If the source-CBBBs do not satisfy conditions (A) and (B), then stop, declaring that no linearization could be found.

The preceding procedure is validated via the following invariant, which is verified using the transitivity of $\rhd$-priority.

> *If the described procedure succeeds, then every CBBB or sum of CBBBs that is added to list L has $\rhd$-priority over every CBBB in the remnant super-*DAG*.*

L thus converges to a $\rhd$-linearization of $\mathcal{G}''$, whose components are CBBBs or sums thereof.

5 The Benefits of the Sweep Algorithm

Algorithm IC-Sweep advances the basic algorithmic framework of IC-scheduling, as developed in [8, 26], along two major fronts. Most importantly, the algorithm allows us to develop IC-optimal schedules for a large class of DAGs that did not yield to the earlier framework. Less obviously, the algorithm speeds up certain procedures required by the framework. We illustrate the former benefit in Section 5.1 and the latter in Section 5.2.

5.1 DAGs That Can Now Be Scheduled IC Optimally

We illustrate our extended IC-scheduling framework by finding IC-optimal schedules for the three DAGs of Fig. 3. We begin with the following schematic scheduling problem. Let

[9] Note that both conditions, (A) and (B), are checked efficiently using Algorithm IC-Sweep.

us be given a DAG $\mathcal{G}$ with $p+1$ *levels* of nodes; i.e., $N_{\mathcal{G}}$ is the disjoint union $N_0 \cup \cdots \cup N_p$, and, for each arc $(u \to v) \in A_{\mathcal{G}}$, there is an $i \in [0, p-1]$ such that $u \in N_i$ and $v \in N_{i+1}$. Say moreover that for each $i \in [0, p-1]$, the induced subgraph of $\mathcal{G}$ on the node-set $N_i \cup N_{i+1}$ is a sum of CBBBs: $\mathcal{G}_i = \mathcal{B}_{i,1} + \cdots + \mathcal{B}_{i,p_i}$. Say finally that:

- each CBBB $\mathcal{B}_{i,j}$ admits an IC-optimal schedule;
- each sum-DAG $\mathcal{G}_i = \mathcal{B}_{i,1} + \cdots + \mathcal{B}_{i,p_i}$ admits an IC-optimal schedule;
- for all $i \in [0, p-1]$, $j \in [1, p_i]$, and $k \in [1, p_{i+1}]$, we have $\mathcal{B}_{i,j} \rhd \mathcal{B}_{i+1,k}$.

Under these circumstances, our extension of IC-scheduling guarantees the IC optimality of the schedule that executes each sum $\mathcal{G}_1, \ldots, \mathcal{G}_p$ in turn. This optimality is verified via repeated invocation of Corollary 4.1.

We instantiate the preceding schematic scheduling problem with two specific ones. We show how our extension of IC-scheduling finds IC-optimal schedules for the two DAGs on the top level of Fig. 3, neither of which could be scheduled IC optimally within the framework of [26]. While both of these illustrative DAGs are artificial ones, their structures are reminiscent of DAGs arising in actual computations, several of which appear in [9]. Thus these are significant examples of the power added by our extended framework.

1. We turn first to the intransigent top-lefthand DAG $\mathcal{G}$ of Fig. 3. We begin by parsing $\mathcal{G}$, using the algorithm of [26], into its seven constituent CBBBs, call them $\mathcal{B}_1$, ..., $\mathcal{B}_7$. The

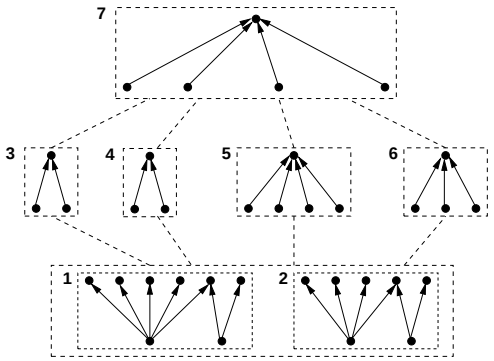

Figure 6: *The super-*DAG *formed from the* DAG $\mathcal{G}$ *of Fig. 3 using the new algorithm.*

resulting super-DAG appears in Fig. 6, with each CBBB $\mathcal{B}_i$ in a dashed box labeled i. Next, we test all inter-CBBB $\triangleright$-priorities. We noted in Table 2 that neither of $\mathcal{B}_1$ nor $\mathcal{B}_2$ has $\triangleright$-priority over the other; but each other pairing of the CBBBs does admit a $\triangleright$-priority, as indicated by the highlighted entries in the subtables of Table 5.

$\mathcal{B}_1$ $(\mathcal{B}_3 = \mathcal{B}_4) \rightarrow$ $\downarrow$	0	1	2
0	**0**	0	1
1	**4**	4	5
2	**6**	**6**	**7**

$\mathcal{B}_2$ $(\mathcal{B}_3 = \mathcal{B}_4) \rightarrow$ $\downarrow$	0	1	2
0	**0**	0	1
1	**3**	3	4
2	**5**	**5**	**6**

$\mathcal{B}_4$ $\mathcal{B}_6 \rightarrow$ $\downarrow$	0	1	2	3
0	**0**	0	0	1
1	**0**	0	0	1
2	**1**	**1**	**1**	**2**

$\mathcal{B}_6$ $\mathcal{B}_5 \rightarrow$ $\downarrow$	0	1	2	3	4
0	**0**	0	0	0	1
1	**0**	0	0	0	1
2	**0**	0	0	0	1
3	**1**	**1**	**1**	**1**	**2**

Table 5: *The Tables $\mathcal{E}$ constructed by Algorithm* 2-IC-Sweep *on pairings of the constituent CBBBs of the* DAG $\mathcal{G}$ *of Fig. 3.*

Specifically, the subtables in Table 5 show explicitly that:

$$[\mathcal{B}_1 \triangleright (\mathcal{B}_3 = \mathcal{B}_4)] \text{ and } [\mathcal{B}_2 \triangleright (\mathcal{B}_3 = \mathcal{B}_4)] \text{ and } [(\mathcal{B}_3 = \mathcal{B}_4) \triangleright \mathcal{B}_6 \triangleright (\mathcal{B}_5 = \mathcal{B}_7)].$$

By transitivity, then, for all $i \in [3, 6]$, $\mathcal{B}_1 \triangleright \mathcal{B}_i$ and $\mathcal{B}_2 \triangleright \mathcal{B}_i$. Therefore, by Theorem 4.1:

$$(\mathcal{B}_1 + \mathcal{B}_2) \ \triangleright \ \mathcal{B}_3 \ \triangleright \ \mathcal{B}_4 \ \triangleright \ \mathcal{B}_6 \ \triangleright \ \mathcal{B}_5 \ \triangleright \ \mathcal{B}_7.$$

These priorities collectively yield an IC-optimal schedule for $\mathcal{G}$, using Theorem 2.1.

2. We next consider the top-righthand DAG $\mathcal{G}'$ of Fig. 3. Having discussed the preceding DAG $\mathcal{G}$ in detail, the reader should be prepared for us to be a bit sketchier with $\mathcal{G}'$. We

first parse $\mathcal{G}'$, using the algorithm of [26], into its three constituent CBBBs, $\mathcal{B}_0$, $\mathcal{B}_1$, and $\mathcal{B}_2$. ($\mathcal{B}_1$ and $\mathcal{B}_2$ retain their names from Section 3.3.1.) Note that any execution of $\mathcal{G}'$ must begin with its single source, after whose execution, we are left with the chore of executing the sum $\mathcal{B}_1 + \mathcal{B}_2$, which we discuss in detail in Section 3.3.1. The reader can now derive the super-DAG of $\mathcal{G}'$ depicted in Fig. 7, from which we readily derive an IC-optimal schedule for $\mathcal{G}'$: execute the source; then execute the IC-optimal schedule derived from Table 2.

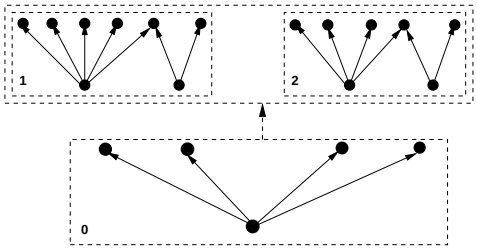

Figure 7: *The super-*DAG *formed from the righthand* DAG $\mathcal{G}'$ *of Fig. 3 via the new algorithm.*

3. We deal finally with a small, but typical, version of the bottom sum-DAG of Fig. 3 depicted in Fig. 8. Easily, the lefthand summand-DAG, call it $\mathcal{G}^{(L)}$ (resp., the righthand

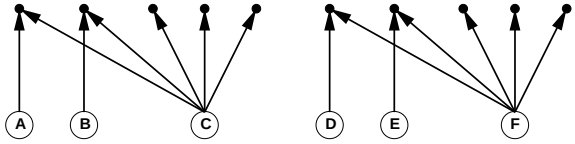

Figure 8: *A small version,* $\mathcal{G}^{(L)} + \mathcal{G}^{(R)}$*, of the fMRI sum-*DAG *at the bottom of Fig. 3.*

summand-DAG, call it $\mathcal{G}^{(R)}$) admits the IC-optimal schedule that executes sources in the order C, B, A (resp., F, D, E) and then executes the sinks in some order that is not relevant to IC quality. Accordingly, we cleave to these orders of source-executions as we use Algorithm 2-IC-Sweep to schedule the sum-DAG $\mathcal{G}^{(L)} + \mathcal{G}^{(R)}$. The result appears in Table 6. The highlighted entries in the table provide several IC-optimal schedules for the sum-DAG, while indicating all of them are interleaved schedules.

$\mathcal{G}^{(L)}$ $\mathcal{G}^{(R)}\rightarrow$ $\downarrow$	0	1	2	3
0	[0]	[3]	4	5
1	[3]	[6]	[7]	[8]
2	4	[7]	[8]	[9]
3	5	[8]	[9]	[10]

Table 6: *The table constructed by Algorithm* 2-IC-Sweep *on the summand* DAG*s of Fig. 8.*

5.2 Speeding Up Scheduling Procedures

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 $\hat{\delta}$-*Planar Bipartite Tree* (*PBT*, for short), denoted $\mathcal{P}[\hat{\delta}]$, is defined inductively as follows; see Fig. 9.

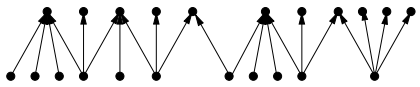

Figure 9: *The PBT* $\mathcal{P}\left[\frac{1}{4}, 3, \frac{1}{3}, 3, \frac{1}{2}, 2, \frac{1}{4}, 3, \frac{1}{2}, 4\right]$.

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

- $\mathcal{P}[d]$ has one source, d sinks, and d arcs connecting the source to each sink.
- $\mathcal{P}\left[\frac{1}{d}\right]$ 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.
- If $\hat{\delta}$ ends with an integer, then $\mathcal{P}\left[\hat{\delta},\ \frac{1}{d}\right]$ 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.

One finds in [8] a recursive algorithm that produces an IC-optimal schedule for any s-source PBT $\mathcal{P}$ in time $\Theta(s^3)$; this timing does not vary with $\mathcal{P}$'s structure. The overall flow of the

algorithm is the repeated discovery, in time $\Theta(s^2)$, for various sub-PBTs of $\mathcal{P}$, of a source that is maximal under a specified partial order. Algorithm IC-Sweep yields an alternative scheduling algorithm for PBTs, that, *in expectation*, is faster by a factor of $O(s)$. This new algorithm, call it $\mathcal{A}$, is also built upon the core task of finding maximal sources of sub-PBTs, but it exploits each PBT's structure to accelerate its search for an IC-optimal schedule. $\mathcal{A}$ proceeds as follows. Let $T(s)$ denote the expected time for $\mathcal{A}$ to find an IC-optimal schedule for PBT $\mathcal{P}$, given that all s-source PBTs are equally likely to occur, so that all of $\mathcal{P}$'s sources are equally likely to be maximal.

1. $\mathcal{A}$ invokes the procedure from [8] to find a maximal source v of $\mathcal{P}$. Via the analysis in that paper, this process takes time $\Theta(s^2)$.

   Because $\mathcal{P}$ is *planar*, if v has outdegree k, then removing it (by executing it) partitions $\mathcal{P}$ into sub-PBTs, $\mathcal{P}_1$ and $\mathcal{P}_2$, having, respectively, $s_1 \in [0, s-1]$ and $s-s_1-1$ sources.
2. In time $T(s_1) + T(s - s_1 - 1)$, $\mathcal{A}$ recursively finds IC-optimal schedules for $\mathcal{P}_1$ and $\mathcal{P}_2$.
3. $\mathcal{A}$ uses Algorithm IC-Sweep to combine these IC-optimal schedules into an IC-optimal schedule for $\mathcal{P}$. By Theorem 3.2, this process takes time $\Theta(s^2)$.

Because expectations add, and because all sources of $\mathcal{P}$ are equally likely to be maximal, we can add up the preceding contributions to $T(s)$. We let $a > 0$ be the constant promised by the big-Θ notation, such that

$$T(s) \;\leq\; \frac{1}{s}\sum_{q=0}^{s-1}\left(T(q) + T(s-q-1)\right) \;+\; as^2 \;\leq\; \frac{2}{s}\sum_{q=0}^{s-1} T(q) + \; as^2. \tag{5.1}$$

Assume, for induction, that there exists a constant $b > 0$ such that $T(m) \leq bm^2$ for all $m < s$. We then have from (5.1):

$$T(s) \;\leq\; \frac{2b}{s}\sum_{q=0}^{s-1} q^2 \;+\; as^2 \;\leq\; \left(a + \frac{2b}{3}\right)s^2 + \text{ l.o.t.}$$

Clearly, if $b > 3a$, then $T(s) \leq bs^2$, which extends the induction.

We thus have a procedure that takes cubic time via a naive implementation, which can be sped up to quadratic expected time by using Algorithm IC-Sweep.

6 Where We Are, and Where We're Going

Conclusions. We have made a major extension to IC Scheduling by devising Algorithm IC-Sweep and incorporating it into the algorithmic scheduling framework of [8, 26]. The thus-extended framework can produce IC-optimal schedules for a much broader range of DAGs, including ones that abstract simplified versions of real scientific computations. Much of the added power of the extension comes from the new ability to craft schedules that *interleave* the execution of sub-DAGs. An additional dividend of Algorithm IC-Sweep, that enhances this extension, is that the algorithm efficiently detects $\triangleright$-priority chains for arbitrary sequences of DAGs and—even more importantly—it decides of DAGs $\mathcal{G}_1, \ldots, \mathcal{G}_p$ that admit IC-optimal schedules, whether or not their sum, $\mathcal{G}_1 + \cdots + \mathcal{G}_p$, admits such a schedule. An unexpected dividend is that Algorithm IC-Sweep sometimes accelerates certain procedures mandated by the framework of [8, 26].

Projections. We are expanding IC Scheduling theory in several directions.

1. 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.
2. We are working to extend our theory so that it can optimally schedule composite DAGs whose building blocks are not necessarily bipartite.

In addition to these theoretical initiatives, we are planning experiments using both WAN simulators and small-scale IC platforms that will help us gauge the implications of our idealized algorithmic framework for real IC.

Acknowledgments. A portion of this research was done at the Univ. of Massachusetts Amherst, while G. Cordasco and G. Malewicz were visiting and A. Rosenberg was on the faculty. The research of G. Malewicz was supported in part by NSF Grant ITR-800864. The research of A. Rosenberg was supported in part by NSF Grants CCF-0342417 and CNS-0842578.

References

[1] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert (2002): Bandwidth-centric allocation of independent tasks on heterogeneous platforms. *Intl. Parallel and Distr. Processing Symp. (IPDPS'02)*.

[2] O. Beaumont, A. Legrand, Y. Robert (2003): The master-slave paradigm with heterogeneous processors. *IEEE Trans. Parallel and Distr. Systs. 14*, 897–908.

[3] M.A. Bender and C.A. Phillips (2007): Scheduling DAGs on asynchronous processors. *19th ACM Symp. on Parallel Algorithms and Architectures*, 35–45.

[4] S.-S. Boutammine, D. Millot, C. Parrot (2006): An adaptive scheduling method for grid computing. *Euro-Par 2006*. In *LNCS 4128*, Springer-Verlag, Berlin, 188–197.

[5] S.-S. Boutammine, D. Millot, C. Parrot (2006): Dynamically scheduling divisible load for grid computing. *High-Performance Computing and Communications*. In *Lecture Notes in Computer Science 4208*, Springer-Verlag, Berlin, 763–772.

[6] R. Buyya, D. Abramson, J. Giddy (2001): A case for economy Grid architecture for service oriented Grid computing. *10th Heterogeneous Computing Wkshp.*

[7] W. Cirne and K. Marzullo (1999): The Computational Co-Op: gathering clusters into a metacomputer. *13th Intl. Parallel Processing Symp.*, 160–166.

[8] G. Cordasco, G. Malewicz, A.L. Rosenberg (2007): Advances in IC-scheduling theory: scheduling expansive and reductive DAGs and scheduling DAGs via duality. *IEEE Trans. Parallel and Distributed Systems 18*, 1607–1617.

[9] G. Cordasco, G. Malewicz, A.L. Rosenberg (2007): Applying IC-scheduling theory to some familiar computations. *Wkshp. on Large-Scale, Volatile Desktop Grids (PCGrid'07).*

[10] G. Cordasco, G. Malewicz, A.L. Rosenberg (2008): Extending IC-scheduling via the Sweep algorithm. *16th Euromicro Conf. on Parallel, Distributed, and Network-Based Processing (PDP'08)*, 366–373.

[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein (1999): *Introduction to Algorithms* (2nd Ed). MIT Press, Cambridge, Mass.

[12] I. Foster and C. Kesselman [eds.] (2004): *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition).* Morgan-Kaufmann, San Francisco.

[13] I. Foster, C. Kesselman, S. Tuecke (2001): The anatomy of the Grid: enabling scalable virtual organizations. *Intl. J. Supercomputer Applications.*

[14] L. Gao and G. Malewicz (2007): Toward maximizing the quality of results of dependent tasks computed unreliably. *Theory of Computing Systs. 41*, 731–752.

[15] L.-X. Gao, A.L. Rosenberg, R.K. Sitaraman (1999): Optimal clustering of tree-sweep computations for high-latency parallel environments. *IEEE Trans. Parallel and Distributed Systems 10*, 813–824.

[16] A. Gerasoulis and T. Yang (1992): A comparison of clustering heuristics for scheduling dags on multiprocessors. *J. Parallel and Distributed Computing 16*, 276–291.

[17] R. Hall, A.L. Rosenberg, A. Venkataramani (2007): A comparison of DAG-scheduling strategies for Internet-based computing. *21st IEEE Intl. Parallel and Distributed Processing Symp.*

[18] H.T. Hsu (1975): An algorithm for finding a minimal equivalent graph of a digraph. *J. ACM 22*, 11–16.

[19] K. Hwang and Z. Xu (1998): *Scalable Parallel Computing: Technology, Architecture, Programming.* McGraw-Hill, New York.

[20] Y.-K. Kwok and I. Ahmad (1999): Benchmarking and comparison of the task graph scheduling algorithms. *J. Parallel and Distributed Computing 59*, 381–422.

[21] Y.-K. Kwok and I. Ahmad (1999): Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys 31*, 406–471.

[22] D. Kondo, H. Casanova, E. Wing, F. Berman (2002): Models and scheduling mechanisms for global computing applications. *Intl. Parallel and Distr. Processing Symp.*

[23] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, M. Lebofsky (2000): SETI@home: massively distributed computing for SETI. In *Computing in Science and Engineering* (P.F. Dubois, Ed.) IEEE Computer Soc. Press, Los Alamitos, CA.

[24] G. Malewicz, I. Foster, A.L. Rosenberg, M. Wilde (2007): A tool for prioritizing DagMan jobs and its evaluation." *J. Grid Computing 5*, 197–212.

[25] G. Malewicz and A.L. Rosenberg (2005): On batch-scheduling DAGs for Internet-based computing. *Euro-Par 2005.* In *Lecture Notes in Computer Science 3648*, Springer-Verlag, Berlin, 262–271.

[26] G. Malewicz, A.L. Rosenberg, M. Yurkewych (2006): Toward a theory for scheduling DAGs in Internet-based computing. *IEEE Trans. Comput. 55*, 757–768.

[27] C.L. McCreary, A.A. Khan, J. Thompson, M. E. Mcardle (1994): A comparison of heuristics for scheduling DAGs on multiprocessors. *8th Intl. Parallel Processing Symp.*, 446–451.

[28] A.L. Rosenberg (2004): On scheduling mesh-structured computations for Internet-based computing. *IEEE Trans. Comput. 53*, 1176–1186.

[29] A.L. Rosenberg and M. Yurkewych (2005): Guidelines for scheduling some common computation-DAGs for Internet-based computing. *IEEE Trans. Comput. 54*, 428–438.

[30] V. Sarkar (1989): *Partitioning and Scheduling Parallel Programs for Multiprocessors.* MIT Press, Cambridge, Mass.

[31] X.-H. Sun and M. Wu (2003): GHS: A performance prediction and node scheduling system for Grid computing. *IEEE Intl. Parallel and Distributed Processing Symp.*

[32] S.W. White and D.C. Torney (1993): Use of a workstation cluster for the physical mapping of chromosomes. *SIAM NEWS*, March, 1993, 14–17.