Extending IC-Scheduling via the Sweep Algorithm

(Extended Abstract)

Gennaro Cordasco[*]
Univ. di Salerno
cordasco@dia.unisa.it

Grzegorz Malewicz[†]
Google Inc.
malewicz@google.com

Arnold L. Rosenberg[‡]
Univ. of Massachusetts.
rsnbrg@cs.umass.edu

Abstract

Earlier work has developed the rudiments of a scheduling theory for computations having intertask dependencies—modeled via dags—for Internet-based computing. The goal of the schedules produced is to render tasks eligible for execution as fast as possible, with the aim of: (a) utilizing clients' computational resources well, by always having work to allocate to an available client; (b) lessening the likelihood of a computation's stalling for lack of eligible tasks. Simulation studies suggest that this goal does accelerate computation over the Internet. The theory crafts a schedule for a dag $\mathcal{G}$ by "parsing" $\mathcal{G}$ (if possible) into connected building-block dags that one can "compose" to form $\mathcal{G}$ and then analyzing the scheduling dependencies among these building blocks. The current paper extends the theory by developing the Sweep Algorithm*, a tool that allows one to: (1) schedule using building blocks that are not necessarily connected, and (2) craft schedules that interleave the execution of subdags that have no interdependencies. The augmented scheduling algorithms allow one to craft optimal schedules for previously unschedulable dags. Examples presented include artificial dags that are "close" to ones arising in real computations, as well as a component of a dag that arises in a functional MRI application.*

1 Introduction

Earlier work [19, 20] has developed *IC-Scheduling*, 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 ([6, 11, 12]), global computing ([7]), and volunteer computing ([15]). The goal is 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 remote clients, by always having work to allocate to an available client; (b) lessening the likelihood of a computation's stalling pending computation of already-allocated tasks. Two simulation studies—[16], which focuses on a small number of dags that arise in real scientific computations, and [13], which derives eligibility-enhancing schedules for hundreds of artificially generated dags—suggest that schedules produced via IC-Scheduling often have marked computational benefits over those produced by a variety of common heuristics (such as FIFO).

Inspired by the case studies of [19, 20], the study in [18] developed IC-Scheduling theory, an algorithmic framework that can optimally schedule a broad class of dags for IC. The development begins with any collection of *building-block dags* that can be scheduled optimally; it introduces two algorithmic notions that allow us to schedule computation-dags built from these building blocks.

1. *The* priority *relation* $\triangleright$ *on dags.* The assertion "$\mathcal{G}_1 \triangleright \mathcal{G}_2$" says that entirely executing first $\mathcal{G}_1$ and then $\mathcal{G}_2$ 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.* If one constructs a computation-dag $\mathcal{G}$ by composing building blocks that are pairwise comparable under relation $\triangleright$, then we can often compute an optimal schedule for $\mathcal{G}$ from optimal schedules for the building blocks.

IC-Scheduling theory is a work in progress. It already optimally schedules dags that arise within a large variety of important computations; Fig. 1 depicts five dags whose optimal schedules are derived in [10, 18, 19, 20]. To illustrate that the theory does not demand the degree of structural uniformity of the dags in Fig. 1, Fig. 2 depicts[1] three artificial dags that the theory also schedules optimally.

[*]Dip. di Informatica e Applicazioni, Univ. di Salerno, Fisciano 84084, ITALY.

[†]Google Inc., Mountain View, CA 94043, USA.

[‡]Dept. of Computer Science, Univ. Massachusetts, Amherst, MA 01003, USA.

[1]Henceforth, graph-edges in figures represent arcs that point upward.

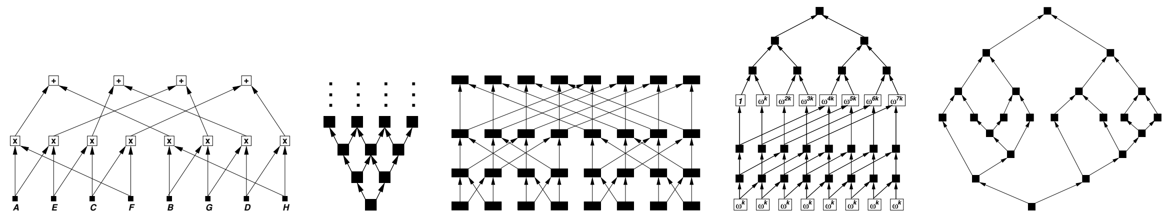

Figure 1. *Data-dependency dags for:* (left to right) *recursive matrix multiplication, a wavefront computation, the FFT; the discrete Laplace transform, a divide-and-conquer computation.*

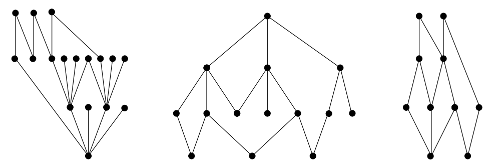

Figure 2. *Three composite dags that the framework of [9, 18] can schedule optimally.*

The past successes of IC-Scheduling theory are tempered by the existence of significant computations that admit optimal schedules but that the current theory cannot schedule optimally. Fig. 3 presents three such dags. The top two

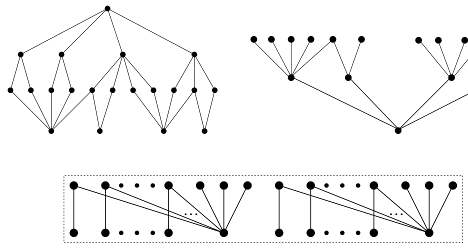

Figure 3. *Three dags that require our extended framework.*

are motivated by their structural similarities to the dags in Figs. 1 and 2. The bottom dag duplicates a subdag of a large functional Magnetic Resonance Imaging computation-dag; see [16] for details. (This subdag appears once in the fMRI dag, but similar subdags appear multiple times, so this duplication retains the spirit of the complete dag.)

The current paper extends IC-Scheduling theory by developing tools (in Sections 3, 4) that significantly expand the repertoire of dags that the theory can schedule optimally. This extension results from algorithms that allow one: (a) to schedule using bipartite building-block dags that are not necessarily connected, (b) to craft schedules that interleave the execution of independent subdags. In particular, the new framework optimally schedules the dags of Fig. 3. The enabling algorithmic tool also efficiently decides $\triangleright$-priority between dags. Importantly, the extended theory can schedule optimally a larger repertoire of real scientific computation-dags; cf. [10].

Related work. Most closely related to our study are its companions in developing IC-Scheduling theory. The topic is introduced in [19, 20], where optimal schedules are produced for several significant dags. The initial algorithmic framework of the theory appears in [18] and is extended significantly in [9], both by allowing one to exploit dag-duality as a scheduling tool and by greatly expanding the repertoire of available building blocks. A companion source, [17], pursues an alternative scheduling regimen for IC, in which a server allocates *batches* of tasks at once. 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, 11, 12, 14, 15, 21].

2 A Basis for a Scheduling Theory

A **(Computation-)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 $(u \to v) \in A_{\mathcal{G}}$: • 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 $(u \to v)$ 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 $A_{\mathcal{G}_1} \cup A_{\mathcal{G}_2}$.

A quality model. When one executes a dag $\mathcal{G}$, each $v \in N_{\mathcal{G}}$ becomes ELIGIBLE (for execution) only after all of its parents have been executed (so sources are always ELI-

GIBLE). We do not allow recomputation, so v loses its ELIGIBILITY once executed. In compensation, executing v may render new nodes ELIGIBLE—when v is their last-executed parent. A *schedule* Σ for $\mathcal{G}$ is a rule for selecting which ELIGIBLE node to execute at each step. Σ's *IC quality* is the number of ELIGIBLE nodes after each execution—the more, the better. (**Note**: *Time is measured in an event-driven manner*, as the number of executed nodes.) Our goal is to execute nodes in an order that maximizes the production rate of ELIGIBLE nodes *at every step of the computation*. If Σ achieves this demanding goal, then it is **IC optimal**. IC optimality has two benefits. Having access to more ELIGIBLE nodes: (1) may reduce the chance of a computation's stalling when remote clients are slow (so that new tasks cannot be allocated pending the return of allocated ones); (2) will allow more (roughly) simultaneous requests for tasks to be satisfied, thereby increasing "parallelism." The simulations in [16, 13] bolster our hope that the preceding intuition does enhance the computational speed under IC.

A Framework for Crafting IC-Optimal Schedules

Lemma 2.1 ([18]) *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 schedule is no less than Σ's.*

For $i = 1, 2$, let the bipartite dag $\mathcal{G}_i$ have s_i sources, and let it admit the IC-optimal schedule Σ_i; moreover, let $E_{\Sigma_i}(t)$ denote the number of ELIGIBLE nonsources on $\mathcal{G}_i$ at step t. If the following inequalities hold:[2]

$$(\forall x \in [0, s_1])\ (\forall y \in [0, s_2]) : \\ E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \ \le \\ E_{\Sigma_1}(\min\{s_1, x+y\}) + E_{\Sigma_2}(\max\{0, x+y-s_1\}), \tag{1}$$

then $\mathcal{G}_1$ *has priority over* $\mathcal{G}_2$, denoted $\mathcal{G}_1 \rhd \mathcal{G}_2$. Informally, one cannot decrease IC quality by executing a source of $\mathcal{G}_1$ whenever possible.

Lemma 2.2 *$\rhd$-priority is transitive [18]. One can decide in time $O(s_1 s_2)$ whether or not $\mathcal{G}_1 \rhd \mathcal{G}_2$.*

On scheduling complex dags. The operation of *composition* is defined inductively as follows.

- Start with a set $\mathcal{B}$ of base dags. (Each base dag in [9, 18] is a *connected bipartite* dag, called a *CBBB*, for "Connected Bipartite Building Block.")
- One composes disjoint dags $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{B}$ to obtain a composite dag $\mathcal{G}$, as follows.
  - $\mathcal{G}$ begins as the *sum*, $\mathcal{G}_1 + \mathcal{G}_2$, with nodes renamed to ensure that $N_\mathcal{G} \cap (N_{\mathcal{G}_1} \cup N_{\mathcal{G}_2}) = \emptyset$.
  - Select some sinks from $\mathcal{G}_1$, and equally many sources from $\mathcal{G}_2$, in the sum $\mathcal{G}_1 + \mathcal{G}_2$.
  - Pairwise identify (i.e., merge) the selected sinks and sources in some way. The merged set of nodes is $N_\mathcal{G}$; the induced set of arcs is $A_\mathcal{G}$.[3]
- Add the dag $\mathcal{G}$ thus obtained to the set $\mathcal{B}$.

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

$\mathcal{G}$ is a *$\rhd$-linear composition* of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ if: • $\mathcal{G}$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$; • each $\mathcal{G}_i \rhd \mathcal{G}_{i+1}$.

Theorem 2.1 ([18]) *Let $\mathcal{G}$ be a $\rhd$-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.*

The algorithms in [18] that determine whether the preceding framework applies to a dag $\mathcal{G}$ operate as follows.

1. $\mathcal{G}$ is "pruned" to remove *shortcuts*, arcs that duplicate existing paths. The resulting "skeleton" dag $\mathcal{G}'$ shares all of its IC-optimal schedules (if any) with $\mathcal{G}$ [18].
2. $\mathcal{G}'$ is "parsed" (when possible) into a collection of CBBBs, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, such that $\mathcal{G}'$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$.
3. $\mathcal{G}$ is replaced by its *super-dag* $\mathcal{G}''$, whose nodes are $\mathcal{G}_1, \ldots, \mathcal{G}_n$, and whose arcs indicate the compositions that created $\mathcal{G}'$. I.e., if $\mathcal{G}'$ was formed by merging some sinks of $\mathcal{G}_i$ with some sources of $\mathcal{G}_j$, then there is an arc from supernode $\mathcal{G}_i$ to supernode $\mathcal{G}_j$ in $\mathcal{G}''$.
4. It is determined whether or not there is an $\rhd$-linearization of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ that is consistent with the topological dependencies within $\mathcal{G}''$; i.e., if $\mathcal{G}_i$ precedes $\mathcal{G}_j$ in a topological sort of $\mathcal{G}''$, then $\mathcal{G}_i \rhd \mathcal{G}_j$ in the $\rhd$-linearization.

The early success of [18] in scheduling significant dags (including those in Fig. 1) leads to the current challenge of expanding the range of dags that we can schedule IC optimally, especially dags that occur in real computations.

3 The IC-Sweep Algorithm

Algorithm IC-Sweep advances IC-Scheduling theory along two axes. Focus on a *sequence* of $p \ge 2$ disjoint dags, $\mathcal{G}_1, \ldots, \mathcal{G}_p$, that all admit IC-optimal schedules.

IC-optimal scheduling. The algorithm either crafts an IC-optimal schedule for $\mathcal{G}_1 + \cdots + \mathcal{G}_p$ or demonstrates that no such schedule exists. Notably, the schedules produced may *interleave* the executions of nonsinks from the $\{\mathcal{G}_i\}$.

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

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

This contrasts with Theorem 2.1, whose schedules execute all nonsinks from each $\mathcal{G}_i$ consecutively.

Deciding $\triangleright$-priorities. The algorithm determines whether or not $\mathcal{G}_1 \triangleright \cdots \triangleright \mathcal{G}_p$.

Timing. The algorithm operates on $\mathcal{G}_1 + \cdots + \mathcal{G}_p$ in time[4]

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

3.1 Algorithm IC-Sweep

Lemma 3.1 *If $\mathcal{G} = \mathcal{G}_1 + \cdots + \mathcal{G}_p$ admits an IC-optimal schedule Σ, then, $(\forall i \in [1,p])$, Σ must execute $\mathcal{G}_i$'s nonsinks within $\mathcal{G}$ in the same order as some IC-optimal schedule for $\mathcal{G}_i$.*

Proof Hint. No order of executing nonsinks from $\mathcal{G}_1$ and from $\mathcal{G}_2$ can beat their IC-optimal schedules' orders. □

Lemma 3.1 suggests the importance of *interleaving* node-executions from the summands. Focus on an IC-optimal schedule Σ_i for each $\mathcal{G}_i$.

3.1.1 Algorithm 2-IC-Sweep is the 2-dag version of Alg. IC-Sweep.

```
Algorithm 2-IC-Sweep
1. Use schedules $\Sigma_1$ and $\Sigma_2$ 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 executing $i$ nonsinks of $\mathcal{G}_1$ and $j$ nonsinks of $\mathcal{G}_2$.
   By Lemma 3.1, we can construct $\mathcal{E}$ as follows:
      $(\forall i \in [0, n_1])(\forall j \in [0, n_2])$
         $\mathcal{E}(i,j) \; = \; E_{\Sigma_1}(i) + E_{\Sigma_2}(j)$.
   /*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 $n_1 \times n_2$ Verification Table $\mathcal{V}$:
   (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 is 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."
```

[4](1) Each $\mathcal{G}_i$ has n_i nonsinks. (2) All variables in asymptotic expressions may grow without bound. (3) Each $E_\Sigma(t)$, the number of ELIGIBLE nodes on $\mathcal{G}$'s *nonsource* nodes at step t, is known in advance.

Theorem 3.1 *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be disjoint dags that, respectively, admit IC-optimal schedules Σ_1 and Σ_2 and have n_1 and n_2 nonsinks. Alg.* 2-IC-Sweep *determines, within time $O(n_1 n_2)$: (**a**) whether or not $\mathcal{G}_1 + \mathcal{G}_2$ admits an IC-optimal schedule, in the positive case providing such a schedule Σ; (**b**) whether or not either $\mathcal{G}_1 \triangleright \mathcal{G}_2$, or $\mathcal{G}_2 \triangleright \mathcal{G}_1$, or both.*

Proof Sketch. Alg. 2-IC-Sweep attempts to maximize the number of ELIGIBLE nodes at every step (thereby producing Σ) by using Σ_i on $\mathcal{G}_i$'s nonsinks ($i = 1, 2$) to construct table $\mathcal{E}$. It seeks a sequence of node-executions that specifies a *rectilinear* path within $\mathcal{E}$—a sequence of downward or rightward entries—that connects $\mathcal{E}(0,0)$ and $\mathcal{E}(n_1, n_2)$ while having each $\mathcal{E}(i,j)$ maximize $\{\mathcal{E}(a,b) \mid a+b = i+j\}$. Any such path of "YES"es in $\mathcal{V}$ specifies an IC-optimal schedule for $\mathcal{G}_1 + \mathcal{G}_2$:
• 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.
The absence of a rectilinear path indicates that one cannot maximize E_Σ at every step.

Further: $\mathcal{G}_1 \triangleright \mathcal{G}_2$ (resp., $\mathcal{G}_2 \triangleright \mathcal{G}_1$) if, and only if, $\mathcal{V}$ contains a path of "YES"es from $\mathcal{V}(0,0)$ to $\mathcal{V}(n_1, n_2)$, that is shaped like uppercase "L," n_1 downward moves followed by n_2 rightward moves (resp., shaped like the digit "7").

The algorithm spends time $O(1)$ at each entry of $\mathcal{V}$. □

Note. *All IC-optimal schedules for a dag produce the same numbers of* ELIGIBLE *nodes at each step; therefore, Alg.* 2-IC-Sweep *produces the same tables, hence makes the same decisions in the same time, no matter which IC-optimal schedules it uses for $\mathcal{G}_1$ and $\mathcal{G}_2$.*

3.1.2 Sweeping multiple dags. Naively extending Alg. 2-IC-Sweep to $p > 2$ dags, $\mathcal{G} = \mathcal{G}_1 + \cdots + \mathcal{G}_p$ by extending $\mathcal{E}$ and $\mathcal{V}$ to p-dimensional tables leads to a time-complexity of $\Omega(n_1 \cdots n_p)$. We increase efficiency by grouping $\mathcal{G}$ in the form $(\cdots(\mathcal{G}_1 + \mathcal{G}_2) + \cdots + \mathcal{G}_p)$, thereby using Alg. 2-IC-Sweep iteratively and achieving the (generally) lower time-complexity (2).

```
Algorithm IC-Sweep
1. Perform Alg. 2-IC-Sweep on the sum $\mathcal{G}_1 + \mathcal{G}_2$.
```

$\mathcal{E}$	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.

2. For each step $k = 2, 3, \ldots$:
   (a) **if** Step $k-1$ does not succeed—i.e., the $(k-1)$th invocation of Alg. 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 Alg. 2-IC-Sweep halts with the answer "YES"—**then** perform Alg. 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 dags, each $\mathcal{G}_i$ admitting the IC-optimal schedule Σ_i and having n_i nonsinks. Alg.* IC-Sweep *determines, within time (2): (**a**) whether or not $\mathcal{G} \overset{def}{=} \mathcal{G}_1 + \cdots + \mathcal{G}_p$ admits an IC-optimal schedule, in the positive case providing such a schedule Σ; (**b**) whether or not $\mathcal{G}_1 \rhd \cdots \rhd \mathcal{G}_p$.*

Proof Sketch. Let $\mathcal{G}_{1,1} = \mathcal{G}_1$, and, inductively, $\mathcal{G}_{1,k} = \mathcal{G}_{1,k-1} + \mathcal{G}_k$. Alg. IC-Sweep processes, in turn, $\mathcal{G}_1 + \mathcal{G}_2$, $\mathcal{G}_{1,2} + \mathcal{G}_3$, ..., $\mathcal{G} = \mathcal{G}_{1,p-1} + \mathcal{G}_p$. Focus on the p-dimensional analogue, $\mathcal{E}_p$, of table $\mathcal{E}$, wherein each

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

1. IC-optimality. Alg. IC-Sweep decides whether or not $\mathcal{G}$ admits an IC-optimal schedule, by seeking a rectilinear path in $\mathcal{E}_p$ whose jth element is maximum over all $\mathcal{E}_p(i_1, \ldots, i_p)$ with $i_1 + \cdots + i_p = j$. Validation is by induction on p (Theorem 3.1 is the base case), after we associate $\mathcal{E}_p$ with the 2-dimensional tables $\mathcal{E}$ that Alg. 2-IC-Sweep creates for the sum $\mathcal{G}_{1,p-1} + \mathcal{G}_p$.

2. Priorities. $\mathcal{G}_1 \rhd \cdots \rhd \mathcal{G}_p$ iff $\mathcal{V}_p$ (constructed in the natural way from $\mathcal{E}_p$) contains a path of "YES"es between $\mathcal{V}_p(0, \ldots, 0)$ and $\mathcal{V}_p(n_1, \ldots, n_{k+1})$ that, dimension-wise, covers, in turn:

$$\begin{aligned}&\mathcal{V}_p(0,0,\ldots,0,0), \ldots, \mathcal{V}_p(n_1,0,\ldots,0,0), \ldots,\\ &\mathcal{V}_p(n_1,n_2,\ldots,n_{p-1},0), \ldots, \mathcal{V}_p(n_1,n_2,\ldots,n_{p-1},n_p).\end{aligned}$$

For efficiency, Alg. IC-Sweep processes $\mathcal{V}_p$ via a sequence of 2-dimensional tables $\mathcal{V}_2^{(2)}, \ldots, \mathcal{V}_2^{(p)}$.

3. Bound (2) holds because sweep-analyzing $\mathcal{G}_1$ and $\mathcal{G}_2$ takes time $T_{1,2} \leq \alpha n_1 n_2$, for some $\alpha > 0$.

Note. *This analysis is independent of the order in which the algorithm orders the $\{\mathcal{G}_i\}$.* □

3.2 Sample Applications of Alg. IC-Sweep

3.2.1 IC optimality via interleaving. Alg. IC-Sweep produces IC-optimal schedules that the theory of [9, 18] cannot. Table 2(left) exposes[5] the following facts about the CBBBs of Fig. 4. (**a**) Neither $\mathcal{B}_1 \rhd \mathcal{B}_2$, nor $\mathcal{B}_2 \rhd \mathcal{B}_1$. (**b**) $\mathcal{B}_1 + \mathcal{B}_2$ admits an IC-optimal schedule. (**c**) Both IC-optimal schedules for $\mathcal{B}_1 + \mathcal{B}_2$ involve interleaving.

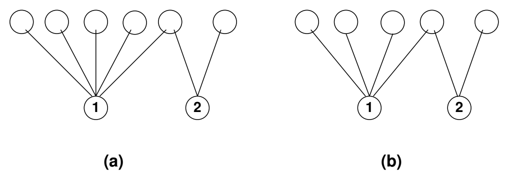

Figure 4. *Two CBBBs whose sum requires an interleaved schedule:* (a) $\mathcal{B}_1$; (b) $\mathcal{B}_2$.

3.2.2 Dags with no IC-optimal schedules. The dag $\mathcal{G}_3$ of Fig. 5(a) does not admit an IC-optimal schedule. To wit,

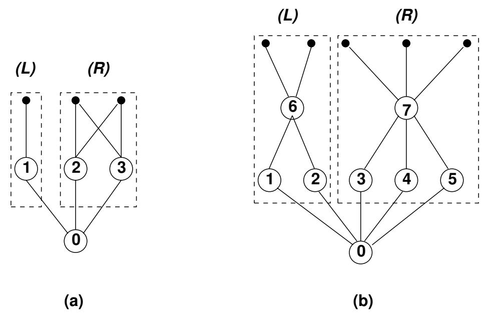

Figure 5. *Two dags that cannot be scheduled IC optimally:* (a) $\mathcal{G}_3$; (b) $\mathcal{G}_4$.

such a schedule would execute $\mathcal{G}_3^{(L)} + \mathcal{G}_3^{(R)}$ IC optimally, where $\mathcal{G}_3^{(L)}$ (resp., $\mathcal{G}_3^{(R)}$) is the subdag of $\mathcal{G}_3$ in the lefthand (resp., righthand) dashed box of Fig. 5(a). Table 2(center) shows that this is impossible. A more complicated, but similar, argument shows that the dag $\mathcal{G}_4$ of Fig. 5(b) does not admit an IC-optimal schedule; cf. Table 2(right).

[5]In all tables, boxed entries indicate maximum entries along diagonals.

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

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

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

Table 2. *The Tables $\mathcal{E}$ for:* (left to right) *the CBBBs of Fig. 4; the left and right boxed subdags of: the dag $\mathcal{G}_3$ of Fig. 5(a); the dag $\mathcal{G}_4$ of Fig. 5(b).*

4 Scheduling Dags Using Alg. IC-Sweep

By enabling interleaved schedules for sums of CBBBs, Alg. IC-Sweep is a major step toward liberating IC-Scheduling from depending on *connected* building blocks.

4.1 The Enabling Theorem

Theorem 4.1 *Let us be given $p+1$ dags $\mathcal{G}_1, \ldots, \mathcal{G}_p$, and $\mathcal{G}'$ such that: • each $\mathcal{G}_i$ 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 $\mathcal{G} \overset{\text{def}}{=} \mathcal{G}_1 + \cdots + \mathcal{G}_p$ admits the IC-optimal schedule Σ. If $\mathcal{G}_i \rhd \mathcal{G}'$ for all $i \in [1,p]$, then $\mathcal{G} \rhd \mathcal{G}'$.*

Proof Hint. The proof is by iterated application of the system (1) that defines $\rhd$-priority. □

Because $\rhd$ is transitive, our expanded scheduling algorithm subsumes the algorithm of [18]:

Corollary 4.1 *Say that each of the $p+q$ dags $\mathcal{G}_1, \ldots, \mathcal{G}_p, \mathcal{G}'_1, \ldots, \mathcal{G}'_q$ admits an IC-optimal schedule. If $\mathcal{G}_1 \rhd \cdots \rhd \mathcal{G}_p \rhd \mathcal{G}'_1 \rhd \cdots \rhd \mathcal{G}'_q$, then $(\mathcal{G}_1 + \cdots + \mathcal{G}_p) \rhd (\mathcal{G}'_1 + \cdots + \mathcal{G}'_q)$.*

4.2 The Consequences of Theorem 4.1

Alg. IC-Sweep enables the following expansion of the algorithmic suite of Section 2.

1. Invoke the first three algorithms of Section 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 with an IC-optimal schedule (if possible),
   (c) the super-dag $\mathcal{G}''$ with node-set $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$, whose arcs indicate compositions thereof.
2. If $\mathcal{G}''$ cannot be generated—because some subalgorithm or condition fails—then the new strategy does not work with $\mathcal{G}$.
3. We seek a $\rhd$-linearization of $\mathcal{G}''$ via CBBBs or sums thereof. We start with:

   • $\mathcal{G}''$, as the *current remnant super-dag* $\mathcal{R}$,
   • an empty list L, as our current progress toward a $\rhd$-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 $\mathcal{R}_{\mathcal{B}_i}$, $\mathcal{B}_i \rhd \mathcal{B}_j$, 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 $\rhd$-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 *not* $\mathcal{B}_i \rhd \mathcal{B}_j$. 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) for all $i \in [1,k]$, and for each source-CBBB $\mathcal{B}_j$ of $\mathcal{R}_{\mathcal{B}'}$ $\mathcal{B}'_i \rhd \mathcal{B}_j$.[6] Then, invoking Theorem 4.1, we append $\mathcal{B}'_1 + \cdots + \mathcal{B}'_k$ to L and 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.

Our procedure is validated via the following invariant, which follows from the transitivity of $\rhd$.

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 ends up as a $\rhd$-linearization of $\mathcal{G}''$, whose components are either CBBBs or sums thereof.

[6]Note that Alg. IC-Sweep checks conditions (A) and (B) efficiently.

5 The Benefits of Algorithm IC-Sweep

Alg. IC-Sweep schedules IC-optimally dags that do not yield to the framework of [9, 18]. In the full paper, we show that it also significantly speeds up certain procedures required by that framework.

Focus on a dag $\mathcal{G}$ with $p+1$ *levels:* $N_{\mathcal{G}}$ is the disjoint union $N_0 \cup \cdots \cup N_p$; each arc of $\mathcal{G}$ goes from some N_i to N_{i+1}. Say that the induced subgraph of $\mathcal{G}$ on each $N_i \cup N_{i+1}$ 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 $\mathcal{G}_i$ admits an IC-optimal schedule; • $(\forall i \in [0, p-1])(\forall j \in [1, p_i])(\forall k \in [1, p_{i+1}])$ $[\mathcal{B}_{i,j} \rhd \mathcal{B}_{i+1,k}]$. Corollary 4.1 implies the IC optimality of executing each sum $\mathcal{G}_1, \ldots, \mathcal{G}_p$ in turn.

We instantiate our schematic scheduling problem with the dags of Fig. 3, finding an IC-optimal schedule for each. These are artificial dags, but they are similar to (sub)dags arising in actual computations; cf. [10, 16]. Thus, they do illustrate the power added by our extended framework.

1. We parse the top-lefthand dag $\mathcal{G}$ of Fig. 3 (via the algorithm of [18]) into CBBBs $\mathcal{B}_1, \ldots, \mathcal{B}_7$. Fig. 6 shows the

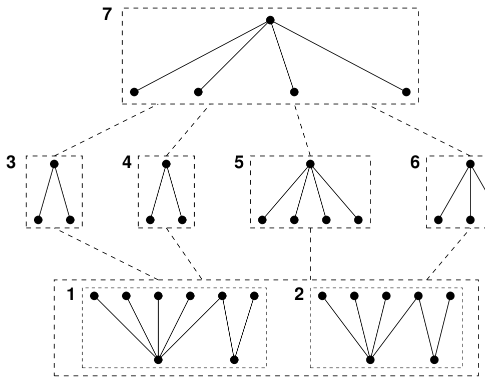

Figure 6. *The super-dag formed from the dag $\mathcal{G}$ of Fig. 3 using the new algorithm.*

resulting super-dag, with each $\mathcal{B}_i$ in a dashed box labeled i. Next, we test all inter-CBBB $\rhd$-priorities. Table 2 shows that neither $\mathcal{B}_1 \rhd \mathcal{B}_2$ nor $\mathcal{B}_2 \rhd \mathcal{B}_1$; but Table 3 shows that all other CBBB pairings do admit a $\rhd$-priority. Specifically:

$$\begin{array}{lll} [\mathcal{B}_1 \rhd (\mathcal{B}_3 = \mathcal{B}_4)] & \text{and} & [\mathcal{B}_2 \rhd (\mathcal{B}_3 = \mathcal{B}_4)] \\ & \text{and} & [(\mathcal{B}_3 = \mathcal{B}_4) \rhd \mathcal{B}_6 \rhd (\mathcal{B}_5 = \mathcal{B}_7)]. \end{array}$$

By transitivity, then, $(\mathcal{B}_1 + \mathcal{B}_2) \rhd \mathcal{B}_3 \rhd \mathcal{B}_4 \rhd \mathcal{B}_6 \rhd \mathcal{B}_5 \rhd \mathcal{B}_7$, which yields an IC-optimal schedule for $\mathcal{G}$.

2. We parse the top-righthand dag $\mathcal{G}'$ of Fig. 3 into three CBBBs: $\mathcal{B}_0$ and $\mathcal{B}_1, \mathcal{B}_2$ from Section 3.2.1. We then derive from the super-dag of Fig. 7 an IC-optimal schedule for $\mathcal{G}'$:

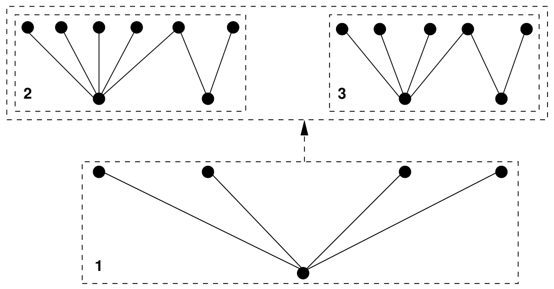

Figure 7. *The super-dag formed from the righthand dag $\mathcal{G}'$ of Fig. 3 using the new algorithm.*

(1) execute the source; (2) execute the IC-optimal schedule from Table 2.

3. Consider finally the bottom dag of Fig. 3; cf. Fig. 8. Using the obvious right-to-left IC-optimal schedules of

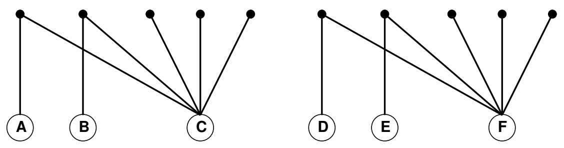

Figure 8. *A small version of the fMRI dag of Fig. 3.*

$\mathcal{G}^{(L)}$ and $\mathcal{G}^{(R)}$, we use see Table 3 to schedule their sum.

These IC-optimal schedules all interleave execution of the summands' sources.

6 Where We Are, and Where We're Going

Conclusions. By incorporating Alg. IC-Sweep into the framework of [9, 18], we can now schedule IC-optimally a much broader range of dags. Much of this new power comes from the ability to *interleave* the execution of subdags. A major dividend is that the Algorithm efficiently decides $\rhd$-priorities for arbitrary sets of dags and, more importantly, decides for dags $\mathcal{G}_1$ and $\mathcal{G}_2$ that admit IC-optimal schedules, whether or not $\mathcal{G}_1 + \mathcal{G}_2$ admits such a schedule. Finally, the Algorithm sometimes accelerates procedures required by the framework of [9, 18].
Projections. We are expanding IC-Scheduling in several directions, most importantly, by seeking a rigorous framework for devising "approximately" IC-optimal schedules. This quest is important because a computationally simple heuristic schedule may be "almost as good" as a more arduously derived IC-optimal one.

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

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

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

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

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

Table 3. *The Tables $\mathcal{E}$ for pairings of the constituent CBBBs of the dag $\mathcal{G}$ of Fig. 3 (top three and left bottom); The Table $\mathcal{E}$ for the summand dags of Fig. 8 (right bottom).*

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 SPAA*, 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 *LNCS 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 HCW.*

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

[8] Condor Project, University of Wisconsin. `http://www.cs.wisc.edu/condor`

[9] 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*, 1607–1617.

[10] G. Cordasco, G. Malewicz, A.L. Rosenberg (2007): Applying IC-scheduling theory to some familiar computations. *PCGrid'07.*

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

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

[13] R. Hall, A.L. Rosenberg, A. Venkataramani (2007): A comparison of dag-scheduling strategies for Internet-based computing. *21st IPDPS.*

[14] D. Kondo, H. Casanova, E. Wing, F. Berman (2002): Models and scheduling mechanisms for global computing applications. *16th IPDPS.*

[15] 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.

[16] 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.

[17] G. Malewicz and A.L. Rosenberg (2005): On batch-scheduling dags for Internet-based computing. *Euro-Par 2005*. In *LNCS 3648*, Springer-Verlag, Berlin, 262–271.

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

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

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

[21] X.-H. Sun and M. Wu (2003): GHS: A performance prediction and node scheduling system for Grid computing. *17th IPDPS.*