

Area-Maximizing Schedules for Series-Parallel DAGs

Gennaro Cordasco¹ and Arnold L. Rosenberg²

¹ University of Salerno,
ISISLab, Dipartimento di Informatica ed Applicazioni “R.M. Capocelli”,
Fisciano 84084, ITALY
cordasco@dia.unisa.it

² Colorado State University,
Department of Electrical and Computer Engineering,
Fort Collins, CO 81523, USA
rsnbrg@colostate.edu

Abstract. Earlier work has introduced a new optimization goal for DAG schedules: the “AREA” of the schedule. AREA-maximizing schedules are intended for computational environments—such as Internet-based computing (IC for short) and massively multicore computers—that would benefit from DAG-schedules that produce execution-eligible tasks as fast as possible. The earlier study of AREA-maximizing schedules showed how to craft such schedules efficiently for DAGs that have the structure of trees, as well as other, less well-known, families of DAGs. The current paper extends the earlier work by showing how to efficiently craft AREA-maximizing schedules for *series-parallel* DAGs, a family that arises, e.g., in multi-threaded computations. The tools that produce the schedules for series-parallel DAGs promise to apply also to other large families of computationally significant DAGs.

1 Introduction

Many of the newest computing platforms, such as the Internet and aggressively multicore architectures, have characteristics that are not addressed by traditional strategies³ for scheduling DAGs (*Directed Acyclic Graphs*), i.e., computations having inter-task dependencies that constrain the order of executing tasks. This issue is discussed at length in, e.g., [24], where the seeds of the *Internet-based computing scheduling*, henceforth IC-scheduling, paradigm are planted. IC-scheduling strives to meet the needs of the new platforms by crafting schedules that execute DAGs in a manner that *maximizes the rate at which new tasks are rendered eligible for execution*. The two aims of the paradigm are: (a) to enhance the effective utilization of available computational resources, by always having work to allocate to an available client/processor; (b) to lessen the likelihood of a computation’s stalling pending completion of already-allocated tasks. Significant progress in [5, 6, 8, 21, 24, 25] has extended the capabilities of IC-scheduling so that it can now optimally schedule a wide range of computationally significant DAGs (cf. [4]). Moreover, simulations using DAGs that arise in real scientific computations [20] as well as structurally similar artificial DAGs [13] suggest that IC-schedules can have substantial computational benefits over schedules produced by a range of common heuristics. However, it has been known since [21] that many significant classes of DAGs do not admit schedules that are optimal within the framework of

³ Many traditional DAG-scheduling strategies are discussed and compared in [12, 18, 22].

IC-scheduling. The current authors have responded to this fact with a relaxed version of IC-scheduling under which *every DAG* admits an optimal schedule [7]. This relaxed strategy strives to maximize the *average* rate at which new tasks are rendered eligible for execution. For reasons that become clear in Section 2, we call the new optimization metric for schedules the *AREA* of the schedule: the goal is an *AREA-maximizing schedule* for a DAG. The study in [7] derived many basic properties of AREA-maximizing schedules. Notable among these are: (1) Every DAG admits an AREA-maximizing schedule. (2) Area-maximization subsumes the goal of IC-scheduling, in the following sense. If a DAG \mathcal{G} admits an optimal IC-schedule, then: (a) that schedule is AREA-maximizing; (b) every AREA-maximizing schedule is optimal under IC-scheduling. Thus, we never lose scheduling quality by focusing on Area-maximization, rather than on achieving an optimal IC-schedule.

Our Contribution. In terms of scheduling algorithmics, the major results of [7] show how to craft AREA-maximizing schedules efficiently for any DAG that has the structure of a *monotonic tree*, as well as for other, less common, families of DAGs. The current paper extends this algorithmic contribution by showing how to efficiently craft an AREA-maximizing schedule for any *series-parallel DAG (SP-DAG, for short)*. SP-DAGs have a regularity of structure that makes them algorithmically advantageous in a broad range of applications; cf. [15, 23, 27]. Most relevant to our study: (1) SP-DAGs are the natural abstraction of many significant classes of computations, including divide-and-conquer algorithms. (2) SP-DAGs admit efficient schedules in parallel computing systems such as CILK [1] that employ a multi-threaded computational paradigm. (3) Arbitrary computation-DAGs can be efficiently (in linear time) reformulated as SP-DAGs with little loss in their degree of parallelism [15, 23]. Our contribution addresses two facets of the existing work on DAG-scheduling: (1) The work on CILK and kindred multi-threaded systems [1–3] employ performance metrics that are not relevant in the computational environments that we target (as discussed earlier). (2) Many SP-DAGs do not admit optimal schedules under IC-scheduling [6, 21].

Related Work. Problems related to scheduling DAGs on parallel/distributed computing systems have been studied for decades. Most versions of these problems are known to be **NP-Hard** [10], except when scheduling special classes of DAGs (cf. [11]). This has led both researchers and practitioners to seek efficient scheduling heuristics that seem to perform well in practice (cf. [14, 26]). Among the interesting attempts to understand such heuristics are the comparisons in [12, 18, 22] and the taxonomy in [19]. Despite the disparate approaches to scheduling in sources such as those cited, virtually every algorithm/heuristic that predates IC-scheduling shared one central characteristic: they all rely on knowing (almost) exact times for each computer to execute each sub-computation and to communicate its results with a collaborating computer. The central premise underlying IC-scheduling (and AREA-maximizing scheduling) is that within many modern computing environments, one cannot even approach such knowledge. This premise is shared by sources such as [16, 17], whose approaches to scheduling for IC platforms admit margins of error in time estimates of 50% or more. Indeed, IC-scheduling and AREA-maximizing scheduling are analytical proposals for what to do when accurate estimates are out of the question.

2 Background

Computation-DAGs and schedules. We study computations that are described by DAGs. Each DAG \mathcal{G} has a set $V_{\mathcal{G}}$ of *nodes*, each representing a *task*, and a set $A_{\mathcal{G}}$ of (directed) *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} .

A parentless node is a *source*; a childless node is a *target*. \mathcal{G} is *connected* if it is so when one ignores arc orientations. When $V_{\mathcal{G}_1} \cap V_{\mathcal{G}_2} = \emptyset$, the *sum* $\mathcal{G}_1 + \mathcal{G}_2$ of DAGs \mathcal{G}_1 and \mathcal{G}_2 is the DAG with node-set $V_{\mathcal{G}_1} \cup V_{\mathcal{G}_2}$ and arc-set $A_{\mathcal{G}_1} \cup A_{\mathcal{G}_2}$.

When one executes a DAG \mathcal{G} , a node $v \in V_{\mathcal{G}}$ becomes ELIGIBLE (for execution) only after all of its parents have been executed. Note that all of \mathcal{G} 's sources are ELIGIBLE at the beginning of an execution; the goal is to render all of \mathcal{G} 's targets ELIGIBLE. Informally, a *schedule* Σ for \mathcal{G} is a rule for selecting which ELIGIBLE node to execute at each step of an execution of \mathcal{G} ; formally, Σ is a *topological sort* of \mathcal{G} , i.e., a linearization of $V_{\mathcal{G}}$ under which all arcs point from left to right (cf. [9]). We do not allow recomputation of nodes/tasks, so a node loses its ELIGIBLE status once it is executed. In compensation, after $v \in V_{\mathcal{G}}$ has been executed, there may be new nodes that are rendered ELIGIBLE; this occurs when v is their last parent to be executed.

We measure the quality of a schedule Σ using the rate at which Σ renders nodes of \mathcal{G} ELIGIBLE. Toward this end, we define⁴, for $k \in [1, |V_{\mathcal{G}}|]$, the quantities $E_{\Sigma}(k)$ and $e_{\Sigma}(k)$: $E_{\Sigma}(k)$ is the number of nodes of \mathcal{G} that are ELIGIBLE after Σ has executed k nodes; and $e_{\Sigma}(k)$ is the number of nodes (perforce, nonsources) of \mathcal{G} that are rendered ELIGIBLE by Σ 's k th node-execution. (We measure time in an event-driven manner, as the number of nodes that have been executed thus far, so we often refer to “step k ” rather than “ Σ 's k th node-execution.”)

The AREA metric for schedules. Focus on a DAG $\mathcal{G} = (V_{\mathcal{G}}, A_{\mathcal{G}})$ with n nontargets, N nonsources, s sources, and S targets (note: $N_{\mathcal{G}} \stackrel{\text{def}}{=} |V_{\mathcal{G}}| = s + N = S + n$). The quality of a schedule Σ for \mathcal{G} at step t is given by the size of $E_{\Sigma}(t)$: the larger, the better. The goal of IC-scheduling 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. A schedule Σ^* that achieves this demanding goal is IC-optimal; formally,

$$(\forall t \in [1, N_{\mathcal{G}}]) E_{\Sigma^*}(t) = \max_{\Sigma \text{ a schedule for } \mathcal{G}} \{E_{\Sigma}(t)\}$$

The AREA of a schedule Σ for \mathcal{G} , $AREA(\Sigma)$, is the sum

$$AREA(\Sigma) \stackrel{\text{def}}{=} E_{\Sigma}(0) + E_{\Sigma}(1) + \cdots + E_{\Sigma}(N_{\mathcal{G}}). \quad (2.1)$$

The normalized AREA, $\widehat{E}(\Sigma) \stackrel{\text{def}}{=} AREA(\Sigma) \div N_{\mathcal{G}}$, is the average number of nodes that are ELIGIBLE when Σ executes \mathcal{G} . The term “*area*” arises by formal analogy with Riemann sums as approximations to integrals. The goal of the scheduling paradigm we develop here is to find an *AREA-maximizing schedule* for \mathcal{G} , i.e., a schedule Σ^* such that

$$AREA(\Sigma^*) = \max_{\Sigma \text{ a schedule for } \mathcal{G}} AREA(\Sigma).$$

⁴ $[a, b]$ denotes the set of integers $\{a, a + 1, \dots, b\}$.

We have discovered in [7] a number of ways to simplify the quest for AREA-maximizing schedules. First, we can restrict the form of such schedules.

Lemma 1. [7] *Altering a schedule Σ for DAG \mathcal{G} so that it executes all of \mathcal{G} 's nontargets before any of its targets cannot decrease Σ 's AREA.*

Hence, we can streamline analysis by ignoring targets. Second, we can alter the AREA metric in certain ways: The only portion of $AREA(\Sigma)$ that actually depends on choices made by Σ is

$$\begin{aligned} area(\Sigma) &\stackrel{\text{def}}{=} \sum_{t=1}^n \sum_{j=1}^t e_{\Sigma}(j) \\ &= ne_{\Sigma}(1) + (n-1)e_{\Sigma}(2) + \dots + e_{\Sigma}(n). \end{aligned} \quad (2.2)$$

The *eligibility profile* associated with schedule Σ for \mathcal{G} is the n -entry vector $\Pi(\Sigma) = \langle e_{\Sigma}(1), e_{\Sigma}(2), \dots, e_{\Sigma}(n) \rangle$.

Our view of schedules as sequences of nontargets nodes allows us to talk about *sub-schedules* of Σ , which are *contiguous subsequences* of Σ . Each subschedule Φ delimits a (not necessarily connected) subDAG \mathcal{G}_{Φ} of \mathcal{G} : $V_{\mathcal{G}_{\Phi}}$ is the subset of $V_{\mathcal{G}}$ whose nodes appear in Φ plus all the nodes in $V_{\mathcal{G}}$ which become ELIGIBLE due to the execution of Φ , and $A_{\mathcal{G}_{\Phi}}$ contains precisely those arcs from $A_{\mathcal{G}}$ that have both ends in $V_{\mathcal{G}_{\Phi}}$.

Let \mathcal{G} be a DAG that admit a schedules Σ . The following abbreviations hopefully enhance legibility. Let Φ be a subschedule of Σ . For any sequence Φ , including (sub)schedules, $\ell_{\Phi} = |\Phi|$ denotes the length of Φ . Note that $E_{\Phi}(k)$ and $e_{\Phi}(k)$ are defined for $k \in [1, \ell_{\Phi}]$.

- $\Pi(\Phi)$ denotes \mathcal{G}_{Φ} 's eligibility profile:

$$\Pi(\Phi) = \langle e_{\Phi}(1), \dots, e_{\Phi}(\ell_{\Phi}) \rangle = \langle e_{\Sigma}(i), \dots, e_{\Sigma}(j) \rangle$$

for some $i, j \in [1, n]$ with $\ell_{\Phi} = j - i + 1$.

- $SUM(\Phi, a, b) = \sum_{i=a}^b e_{\Phi}(i)$, and $SUM(\Phi) = SUM(\Phi, 1, \ell_{\Phi})$.

Let Φ and Ψ be disjoint subschedules of Σ . We denote by $(\Phi \cdot \Psi)$ the subschedule of Σ that concatenates Φ and Ψ . Thus: $(\Phi \cdot \Psi)$ has $\ell_{\Phi} + \ell_{\Psi}$ elements; it first executes all nodes in Φ , in the same order as Φ does, and then executes all nodes in Ψ , in the same order as Ψ does.

Series-parallel DAGs (SP-DAGs). A (*2-terminal*) *series-parallel* DAG \mathcal{G} (*SP-DAG*, for short) is produced by a sequence of the following operations (cf. Fig. 1):

1. **Create.** Form a DAG \mathcal{G} that has:
 - (a) two nodes, a *source* s and a *target* t , which are jointly \mathcal{G} 's *terminals*,
 - (b) one arc, $(s \rightarrow t)$, directed from s to t .
2. **Compose** SP-DAGs, \mathcal{G}' with terminals s', t' , and \mathcal{G}'' , with terminals s'', t'' .
 - (a) **Parallel composition.** Form the SP-DAG $\mathcal{G} = \mathcal{G}' \uparrow \mathcal{G}''$ from \mathcal{G}' and \mathcal{G}'' by identifying/merging s' with s'' to form a new source s and t' with t'' to form a new target t .
 - (b) **Series composition.** Form the SP-DAG $\mathcal{G} = (\mathcal{G}' \rightarrow \mathcal{G}'')$ from \mathcal{G}' and \mathcal{G}'' by identifying/merging t' with s'' . \mathcal{G} has the single source s' and the single target t'' .

One easily uses examples from [6] to craft SP-DAGs that do not admit optimal IC-schedules.

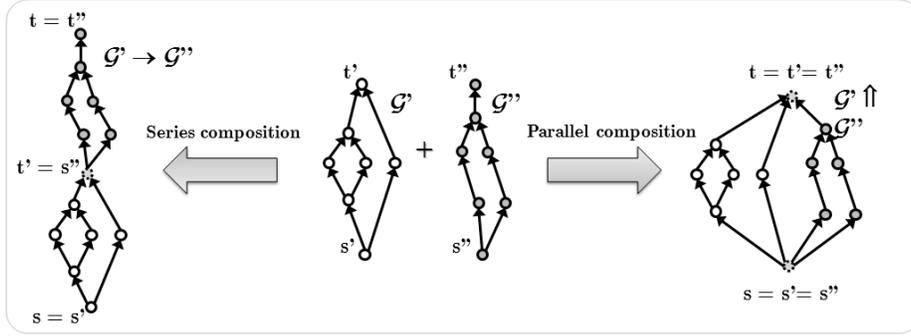


Fig. 1. Compositions of SP-DAGs.

3 Maximizing Area for Series-Parallel DAGs

We turn now to our main result.

Theorem 1. *There exists an algorithm $\mathcal{A}_{\text{SP-DAG}}$ that finds, in time $O(n^2)$, an AREA-maximizing schedule for any n -node SP-DAG.*

The remainder of the section is devoted to proving Theorem 1 by developing Algorithm $\mathcal{A}_{\text{SP-DAG}}$.

3.1 The Idea behind Algorithm $\mathcal{A}_{\text{SP-DAG}}$

The problem of recognizing when a given DAG is an SP-DAG is classic within the area of algorithm design. A decomposition-based algorithm in [27], which solves the problem in linear time, supplies the basic idea underlying Algorithm $\mathcal{A}_{\text{SP-DAG}}$. We illustrate how with the sample SP-DAG \mathcal{G} in Fig. 2(left).

The fact that \mathcal{G} is an SP-DAG can be verified with the help of a binary decomposition tree $\mathcal{T}_{\mathcal{G}}$ whose structure illustrates the sequence of series and parallel compositions that form \mathcal{G} from a set of nodes; hence, $\mathcal{T}_{\mathcal{G}}$ ultimately takes one back to the definition of SP-DAG in Section 2. Fig. 2(right) depicts $\mathcal{T}_{\mathcal{G}}$ for the \mathcal{G} of Fig. 2(left). The leaves of $\mathcal{T}_{\mathcal{G}}$ are single-arc DAGs; each of its internal nodes represents the SP-DAG obtained by composing (in series or in parallel) its two children. Importantly for the design of Algorithm $\mathcal{A}_{\text{SP-DAG}}$, one can use the algorithm of [27] to construct $\mathcal{T}_{\mathcal{G}}$ from a given DAG \mathcal{G} : the construction succeeds just when \mathcal{G} is an SP-DAG. Algorithm $\mathcal{A}_{\text{SP-DAG}}$ uses $\mathcal{T}_{\mathcal{G}}$ to design an AREA-maximizing schedule for \mathcal{G} inductively.

3.2 Algorithm $\mathcal{A}_{\text{SP-DAG}}$'s Inductive Approach

Given an SP-DAG \mathcal{G} , Algorithm $\mathcal{A}_{\text{SP-DAG}}$ first constructs $\mathcal{T}_{\mathcal{G}}$. It then designs an AREA-maximizing schedule for \mathcal{G} by exploiting the structure of $\mathcal{T}_{\mathcal{G}}$.

A. Leaf-DAGs of $\mathcal{T}_{\mathcal{G}}$. Each leaf-DAG is a single-arc DAG, i.e., a series composition of degenerate one-node DAGs. Each such DAG has the form $(s \rightarrow t)$, hence admits a unique schedule (execute s ; then execute t) which, perforce, is AREA-maximizing.

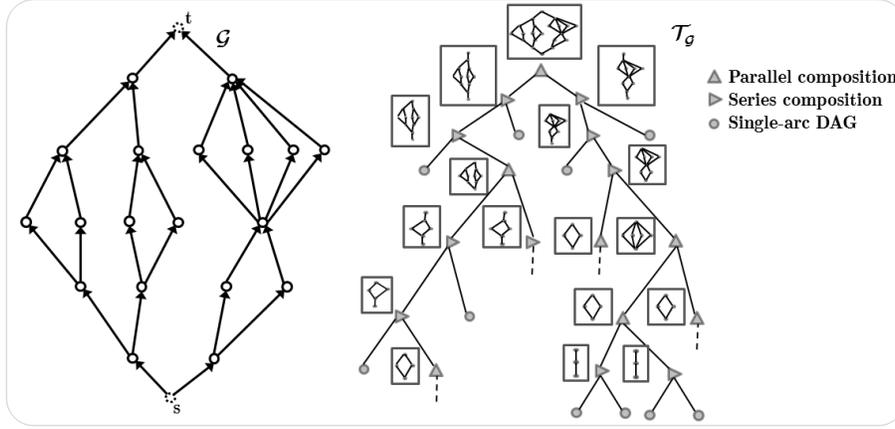


Fig. 2. An example of the decomposition of SP-DAGs.

B. Series-composing internal nodes of \mathcal{T}_G . Focus on a node of \mathcal{T}_G that represents the series composition ($\mathcal{G}' \rightarrow \mathcal{G}''$) of disjoint SP-DAGs \mathcal{G}' and \mathcal{G}'' .

Lemma 2. *If the disjoint SP-DAGs \mathcal{G}' and \mathcal{G}'' admit, respectively, Area-maximizing schedules Σ' and Σ'' , then the schedule $(\Sigma' \cdot \Sigma'')$ is AREA-maximizing for the series composition ($\mathcal{G}' \rightarrow \mathcal{G}''$).*

Proof. Let \mathcal{G}' have N' nonsources, and let \mathcal{G}'' have n'' nontargets. By definition of ELIGIBILITY, every node of \mathcal{G}' must be executed before any node of \mathcal{G}'' , so we need focus only on schedules for $(\mathcal{G}' \rightarrow \mathcal{G}'')$ of the form $(\Sigma_1 \cdot \Sigma_2)$, where Σ_1 is a schedule for \mathcal{G}' and Σ_2 is a schedule for \mathcal{G}'' . We then have

$$\begin{aligned} \text{area}(\Sigma_1 \cdot \Sigma_2) &= \text{area}(\Sigma_1) + \text{area}(\Sigma_2) + \ell_{\Sigma_2} \text{SUM}(\Sigma_1) \\ &= \text{area}(\Sigma_1) + \text{area}(\Sigma_2) + n'' N'. \end{aligned}$$

Thus, choosing schedules Σ_1 and Σ_2 that maximize both $\text{area}(\Sigma_1)$ and $\text{area}(\Sigma_2)$ will maximize $\text{area}(\Sigma_1 \cdot \Sigma_2)$. The lemma follows. \square

C. Parallel-composing internal nodes of \mathcal{T}_G . Focus on a node of \mathcal{T}_G that represents the parallel composition ($\mathcal{G}' \uparrow \mathcal{G}''$) of disjoint SP-DAGs \mathcal{G}' and \mathcal{G}'' . We present an algorithm that uses AREA-maximizing schedules Σ' and Σ'' for \mathcal{G}' and \mathcal{G}'' , respectively, to craft an AREA-maximizing schedule Σ for $(\mathcal{G}' \uparrow \mathcal{G}'')$.

Algorithm SP-AREA

Input: SP-DAGs \mathcal{G}' and \mathcal{G}'' and respective AREA-maximizing schedules Σ' and Σ'' .
Output: AREA-maximizing schedule Σ for $\mathcal{G} = (\mathcal{G}' \uparrow \mathcal{G}'')$.

Note that \mathcal{G}' and \mathcal{G}'' are disjoint within $(\mathcal{G}' \uparrow \mathcal{G}'')$, except for their shared source s and target t . Because every schedule for $(\mathcal{G}' \uparrow \mathcal{G}'')$ must begin by executing s and finish by executing t , we can focus only on how to schedule the *sum*, $\mathcal{G}_1 + \mathcal{G}_2$, obtained by removing both s and t from $(\mathcal{G}' \uparrow \mathcal{G}'')$: \mathcal{G}_1 and \mathcal{G}_2 are not-necessarily-connected subDAGs of, respectively, \mathcal{G}' and \mathcal{G}'' . (The parallel composition in Fig. 1 illustrates

that \mathcal{G}_1 and/or \mathcal{G}_2 can be disconnected: removing s and t in the figure disconnects the lefthand DAG.)

(1) Construct the *average-eligibility profile* $AVG(\Sigma')$ from $\Pi(\Sigma')$ as follows. If $\Pi(\Sigma') = \langle e_{\Sigma'}(1), e_{\Sigma'}(2), \dots, e_{\Sigma'}(n) \rangle$, then

$$AVG(\Sigma') = \langle a_{\Sigma'}(1), a_{\Sigma'}(2), \dots, a_{\Sigma'}(n) \rangle,$$

where, for $k \in [1, n]$, $a_{\Sigma'}(k) = \frac{1}{k} \sum_{i=1}^k e_{\Sigma'}(i)$.

Similarly, construct the average-eligibility profile $AVG(\Sigma'')$ from $\Pi(\Sigma'')$.

(2) (a) Let j' be the smallest index of $AVG(\Sigma')$ whose value is maximum within profile $AVG(\Sigma')$. Segregate the subsequence of Σ' comprising elements $1, \dots, j'$, to form an (*indivisible*) *block* of nodes of Σ' with *average eligibility value* (AEV) $a_{\Sigma'}(j')$. Perform a similar analysis for Σ'' to determine the value j'' and the associated block of nodes of Σ'' with AEV $a_{\Sigma''}(j'')$.

(b) Repeat procedure (a) for Σ' , using indices from $j' + 1$ to n , collecting blocks, until we find a block that ends with the last-executed node of Σ' . Do the analogous repetition for Σ'' .

After procedure (a)-then-(b), each of Σ' and Σ'' is decomposed into a sequence of blocks, plus the associated sequences of AEVs.

We claim that the following schedule Σ for $(\mathcal{G}' \uparrow \mathcal{G}'')$ is AREA-maximizing.

- Σ : 1. Execute s
 2. Merge the blocks of Σ' and Σ'' in nonincreasing order of AEV.
 (Blocks are kept intact; ties in AEV are broken arbitrarily.)
 3. Execute t

Claim. Σ is a valid schedule for $(\mathcal{G}' \uparrow \mathcal{G}'')$.

This is obvious because: (1) Σ keeps the blocks of both Σ' and Σ'' intact, and (2) Σ incorporates the blocks of Σ' (resp., of Σ'') in the same order as they occur in Σ' (resp., in Σ'').

Claim. Σ is an AREA-maximizing schedule for $(\mathcal{G}' \uparrow \mathcal{G}'')$.

Before verifying Σ 's optimality, we digress to establish two technical lemmas.

Two technical lemmas.

Lemma 3. *Let Σ be a schedule for DAG \mathcal{G} , and let Φ and Ψ be disjoint subschedules of Σ .⁵ Then*

$$\left[\text{area}(\Phi \cdot \Psi) > \text{area}(\Psi \cdot \Phi) \right] \text{ iff } \left[\frac{SUM(\Phi)}{l_\Phi} > \frac{SUM(\Psi)}{l_\Psi} \right].$$

Proof. Invoking (2.2), we find that

$$\begin{aligned} \text{area}(\Phi \cdot \Psi) &= (l_\Phi + l_\Psi)e_\Phi(1) + (l_\Phi + l_\Psi - 1)e_\Phi(2) + \dots + (l_\Psi + 1)e_\Phi(l_\Phi) \\ &\quad + l_\Psi e_\Psi(1) + (l_\Psi - 1)e_\Psi(2) + \dots + e_\Psi(l_\Psi); \\ \text{area}(\Psi \cdot \Phi) &= (l_\Phi + l_\Psi)e_\Psi(1) + (l_\Phi + l_\Psi - 1)e_\Psi(2) + \dots + (l_\Phi + 1)e_\Psi(l_\Psi) \\ &\quad + l_\Phi e_\Phi(1) + (l_\Phi - 1)e_\Phi(2) + \dots + e_\Phi(l_\Phi). \end{aligned}$$

Therefore, $\text{area}(\Phi \cdot \Psi) - \text{area}(\Psi \cdot \Phi) = l_\Psi \cdot SUM(\Phi) - l_\Phi \cdot SUM(\Psi)$.

The result now follows by elementary calculation. \square

⁵ The disjointness of Φ and Ψ ensures that both $(\Phi \cdot \Psi)$ and $(\Psi \cdot \Phi)$ are subschedules of Σ .

Lemma 4. *Let Σ be a schedule for DAG \mathcal{G} , and let Φ , Ψ , Γ , and Δ be four mutually disjoint subschedules of Σ . Then*

$$\left[\text{area}(\Phi \cdot \Psi) > \text{area}(\Psi \cdot \Phi) \right] \text{ iff } \left[\text{area}(\Gamma \cdot \Phi \cdot \Psi \cdot \Delta) > \text{area}(\Gamma \cdot \Psi \cdot \Phi \cdot \Delta) \right].$$

The import of Lemma 4 is: *One cannot change the area-ordering of the two concatenations of Φ and Ψ by appending the same fixed subschedule (Γ) before the concatenations and/or appending the same fixed subschedule (Δ) after the concatenations.*

Proof. The following two differences have the same sign:

$$\left[\text{area}(\Phi \cdot \Psi) - \text{area}(\Psi \cdot \Phi) \right] \text{ and } \left[\text{area}(\Gamma \cdot \Phi \cdot \Psi \cdot \Delta) - \text{area}(\Gamma \cdot \Psi \cdot \Phi \cdot \Delta) \right].$$

To wit:

$$\begin{aligned} \text{area}(\Gamma \cdot \Phi \cdot \Psi \cdot \Delta) &= (\ell_\Gamma + \ell_\Phi + \ell_\Psi + \ell_\Delta)e_\Gamma(1) + \cdots + (1 + \ell_\Phi + \ell_\Psi + \ell_\Delta)e_\Gamma(\ell_\Gamma) \\ &\quad + (\ell_\Gamma + \ell_\Phi + \ell_\Delta)e_\Phi(1) + \cdots + (1 + \ell_\Psi + \ell_\Delta)e_\Phi(\ell_\Phi) \\ &\quad + (\ell_\Psi + \ell_\Delta)e_\Psi(1) + \cdots + (1 + \ell_\Delta)e_\Psi(\ell_\Psi) \\ &\quad + \ell_\Delta e_\Delta(1) + \cdots + e_\Delta(\ell_\Delta); \\ \text{area}(\Gamma \cdot \Psi \cdot \Phi \cdot \Delta) &= (\ell_\Gamma + \ell_\Psi + \ell_\Phi + \ell_\Delta)e_\Gamma(1) + \cdots + (1 + \ell_\Psi + \ell_\Phi + \ell_\Delta)e_\Gamma(\ell_\Gamma) \\ &\quad + (\ell_\Psi + \ell_\Phi + \ell_\Delta)e_\Psi(1) + \cdots + (1 + \ell_\Phi + \ell_\Delta)e_\Psi(\ell_\Psi) \\ &\quad + (\ell_\Phi + \ell_\Delta)e_\Phi(1) + \cdots + (1 + \ell_\Delta)e_\Phi(\ell_\Phi) \\ &\quad + \ell_\Delta e_\Delta(1) + \cdots + e_\Delta(\ell_\Delta). \end{aligned}$$

Elementary calculation now shows that

$$\left[\text{area}(\Gamma \cdot \Phi \cdot \Psi \cdot \Delta) - \text{area}(\Gamma \cdot \Psi \cdot \Phi \cdot \Delta) \right] = \left[\text{area}(\Phi \cdot \Psi) - \text{area}(\Psi \cdot \Phi) \right]. \quad \square$$

The optimality of schedule Σ . We focus only on step 2 of Σ because, as noted earlier, every schedule for $(\mathcal{G}' \uparrow \mathcal{G}'')$ begins by executing s and ends by executing t . We validate each of the salient properties of Algorithm SP-AREA.

Lemma 5. *The nodes inside each block determined by Algorithm SP-AREA cannot be rearranged in any AREA-maximizing schedule for $(\mathcal{G}' \uparrow \mathcal{G}'')$.*

Proof. We proceed by induction on the structure of the decomposition tree $\mathcal{T}_\mathcal{G}$ of $\mathcal{G} = (\mathcal{G}' \uparrow \mathcal{G}'')$. The base of the induction is when $\mathcal{T}_\mathcal{G}$ consists of a single leaf-node. The lemma is trivially true in this case, for the structure of the resulting block is mandated by the direction of the arc in the node. (In *any* schedule for the DAG $(s \rightarrow t)$, s must be computed before t .) The lemma is trivially true also when \mathcal{G} is formed solely via parallel compositions. To wit, for any such DAG, removing the source and target leaves one with a set of independent nodes. The resulting eligibility profile is, therefore, a sequence of 0s: each block has size 1 and AEV 0. This means that *every* valid schedule is AREA-maximizing.

In general, as we decompose \mathcal{G} : (a) The subDAG schedule needed to process a *parallel composition* $(\mathcal{G}_1 \uparrow \mathcal{G}_2)$ does not generate any blocks other than those generated by the schedules for \mathcal{G}_1 and \mathcal{G}_2 . (b) The subDAG schedule needed to process a *series*

composition can generate new blocks. To wit, consider SP-DAGs \mathcal{G}_1 and \mathcal{G}_2 , with respective schedules Σ_1 and Σ_2 , so that $(\Sigma_1 \cdot \Sigma_2)$ is a schedule for the serial composition $(\mathcal{G}_1 \rightarrow \mathcal{G}_2)$. Say that $(\Sigma_1 \cdot \Sigma_2)$ generates consecutive blocks, B_1 from Σ_1 and B_2 from Σ_2 , where B_1 's AEV is smaller than B_2 's. Then blocks B_1 and B_2 are merged—via concatenation—by $(\Sigma_1 \cdot \Sigma_2)$ into block $B_1 \cdot B_2$. For instance, the indicated scenario occurs when B_1 is the last block of Σ_1 and B_2 is the first block of Σ_2 . Assuming, for induction, that the nodes in each of B_1 and B_2 cannot be rearranged, we see that the nodes inside $B_1 \cdot B_2$ cannot be reordered. That is, every node of \mathcal{G}_1 —including those in B_1 —must be executed before any node of \mathcal{G}_2 —including those in B_2 . \square

Lemma 5 tells that the nodes inside each block generated by Algorithm SP-AREA cannot be reordered. We show now that blocks can also not be subdivided. Note that this will also preclude merging blocks in ways that violate blocks' indivisibility.

Lemma 6. *If a schedule Σ for a DAG $(\mathcal{G}' \uparrow \mathcal{G}'')$ subdivides a block (as generated by Algorithm SP-AREA), then Σ is not AREA-maximizing.*

Proof. Assume, for contradiction, that there exists an AREA-maximizing schedule Σ for $\mathcal{G} = (\mathcal{G}' \uparrow \mathcal{G}'')$ that subdivides some block A of Σ' into two blocks, B and C . (Our choice of Σ' here clearly loses no generality.) This subdivision means that within Σ , B and C are separated by some sequence D .

We claim that Σ 's AREA-maximality implies that $AEV(C) > AEV(B)$. Indeed, by construction, we know that

$$(\forall j \in [1, \ell_A - 1]) \left[\frac{1}{\ell_A} SUM(A, 1, \ell_A) > \frac{1}{j} SUM(A, 1, j) \right].$$

Therefore, we have

$$\frac{1}{\ell_A} SUM(A, 1, \ell_A) > \frac{1}{\ell_B} SUM(A, 1, \ell_B),$$

which means that

$$\ell_B \sum_{i=1}^{\ell_A} e_A(i) > \ell_A \sum_{i=1}^{\ell_B} e_A(i).$$

It follows, noting that $\ell_C = \ell_A - \ell_B$, that

$$\ell_B \sum_{i=\ell_B+1}^{\ell_A} e_A(i) > (\ell_A - \ell_B) \sum_{i=1}^{\ell_B} e_A(i).$$

This, finally, implies that $AEV(C) > AEV(B)$.

Now we are in trouble: Either of the following inequalities,

$$(a) [AEV(D) < AEV(C)] \quad \text{or} \quad (b) [AEV(D) \geq AEV(C)] \quad (3.3)$$

would allow us to increase Σ 's AREA, thereby contradicting Σ 's alleged AREA-maximality! If inequality (3.3(a)) held, then Lemmas 3 and 4 would allow us to increase Σ 's AREA by interchanging blocks D and C . If inequality (3.3(b)) held, then because $[AEV(C) > AEV(B)]$, we would have $[AEV(D) > AEV(B)]$. But then Lemmas 3 and 4 would allow us to increase Σ 's AREA by interchanging blocks B and D . We conclude that schedule Σ cannot exist. \square

We summarize the results of this section in the following result.

Theorem 2. Let \mathcal{G}' and \mathcal{G}'' be disjoint SP-DAGs that, respectively, admit AREA-maximizing schedules Σ' and Σ'' , and have n' and n'' nontargets. If we know the block decompositions of Σ' and Σ'' , ordered by AEV, then Algorithm SP-AREA determines, within time $O(n' + n'')$ an AREA-maximizing schedule for $\mathcal{G} = (\mathcal{G}' \uparrow \mathcal{G}'')$.

Proof. By Lemma 6, any AREA-maximizing schedule for \mathcal{G} must be a permutation of the blocks obtained by decomposing Σ' and Σ'' . Assume, for contradiction, that some AREA-maximizing schedule Σ^* is not obtained by selecting blocks in decreasing order of AEV. There would then exist (at least) two blocks, A and B , that appear consecutively in Σ^* , such that $AEV(A) < AEV(B)$. We would then have $SUM(A)/\ell_A < SUM(B)/\ell_B$, so by Lemma 3, $area(A \cdot B) > area(B \cdot A)$. Let C (resp., D) be the concatenation of all blocks that come before A (resp., after B) in Σ^* . An invocation of Lemma 4 shows that Σ^* cannot be AREA-maximizing, contrary to assumption.

Timing: Because the blocks obtained by the decomposition are already ordered, completing \mathcal{G} 's schedule requires only merging two ordered sequences of blocks, which can be done in linear time. \square

Observation 1 It is worth noting that Algorithm \mathcal{A}_{SP-DAG} 's decomposition phase, as described in Section 3.2, may require time quadratic in the size of \mathcal{G} . However, we use Algorithm SP-AREA in a recursive manner, so the decomposition for an SP-DAG \mathcal{G} is simplified by the algorithm's (recursive) access to the decompositions of \mathcal{G} 's subDAGs.

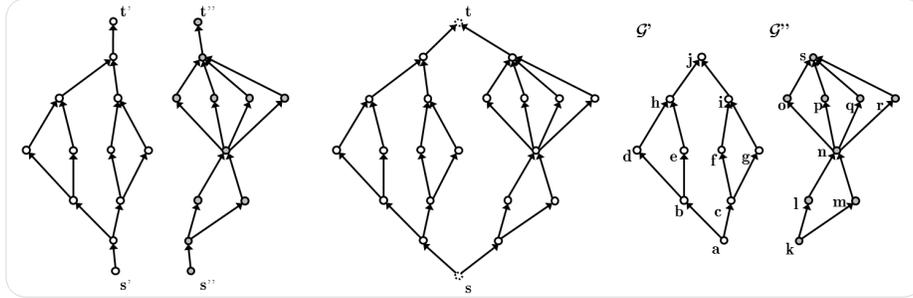


Fig. 3. An example of composition.

An example. The DAG \mathcal{G} of Fig. 3(center) is the parallel composition of the DAGs of Fig. 3(left). (\mathcal{G} appears at the root of $\mathcal{T}_{\mathcal{G}}$ in Fig. 2.) By removing nodes s and t from \mathcal{G} , we obtain the disjoint DAGs \mathcal{G}' and \mathcal{G}'' of Fig. 3(right). We note the AREA-maximizing schedules:

$$\Sigma' = \langle a, b, c, d, e, f, g, h, i \rangle \text{ for } \mathcal{G}'; \quad \Sigma'' = \langle k, l, m, n, o, p, q, r \rangle \text{ for } \mathcal{G}''.$$

We use \mathcal{G}' and \mathcal{G}'' to illustrate Algorithm SP-AREA. For schedule Σ' : $\Pi(\Sigma') = \langle 2, 2, 2, 0, 1, 0, 1, 0, 1 \rangle$, and profile $AVG(\Sigma') = \langle 2, 2, 2, 3/2, 7/5, 7/6, 8/7, 1, 1 \rangle$. The position of the maximum element of $AVG(\Sigma')$ is 3 (choosing the rightmost element in case of ties), so the first block is $\langle a, b, c \rangle$, with AEV 2. Continuing: the new eligibility profile is $\langle 0, 1, 0, 1, 0, 1 \rangle$, the average eligibility profile is $\langle 0, 1/2, 1/3, 1/2, 2/5, 1/2 \rangle$. The maximum is the last element, so the new block is $\langle d, e, f, g, h, i \rangle$, with AEV 1/2. For schedule Σ'' : $\Pi(\Sigma'') = \langle 2, 0, 1, 4, 0, 0, 0, 1 \rangle$, and $AVG(\Sigma'') = \langle 2, 1, 1, 7/4, 7/5, 7/6, 1, 1 \rangle$. The maximum is at the first element, so the first block is $\langle k \rangle$, with AEV

2. The next eligibility profile is $\langle 0, 1, 4, 0, 0, 0, 1 \rangle$, and the average eligibility profile is $\langle 0, 1/2, 5/3, 5/4, 1, 5/6, 1 \rangle$, which has its maximum at the 3rd element. The new block is $\langle l, m, n \rangle$, with AEV $5/3$. The new average eligibility profile is $\langle 0, 0, 0, 1/4 \rangle$, so the last block is $\langle o, p, q, r \rangle$ with AEV $1/4$. Thus, the two schedules are split into five blocks:

Σ' Blocks	AEV
$\langle a, b, c \rangle$	2
$\langle d, e, f, g, h, i \rangle$	1/2

Σ'' Blocks	AEV
$\langle k \rangle$	2
$\langle l, m, n \rangle$	5/3
$\langle o, p, q, r \rangle$	1/4

The AREA-maximizing schedule obtained by ordering the blocks in order of decreasing AEV is $\langle a, b, c, k, l, m, n, d, e, f, g, h, i, o, p, q, r \rangle$.

3.3 The Timing of Algorithm $\mathcal{A}_{\text{SP-DAG}}$

Let \mathcal{G} be an N -node SP-DAG, and let $T(N)$ be the time that Algorithm $\mathcal{A}_{\text{SP-DAG}}$ takes to find an AREA-maximizing schedule Σ for \mathcal{G} , plus the time it takes to decompose Σ into (indivisible) ordered blocks. Let us see what goes into $T(N)$.

- (1) Algorithm $\mathcal{A}_{\text{SP-DAG}}$ invokes [27] to decompose \mathcal{G} in time $O(N)$.
- (2) The algorithm finds an AREA-maximizing schedule Σ for \mathcal{G} by recursively unrolling \mathcal{G} 's decomposition tree $\mathcal{T}_{\mathcal{G}}$.

(2.a) If \mathcal{G} is a series composition ($\mathcal{G}' \rightarrow \mathcal{G}''$) of an n -node SP-DAG \mathcal{G}' and an $(N-n+1)$ -node SP-DAG \mathcal{G}'' , then, by induction, $T(N) = T(N-n+1) + T(n) + O(N)$ for some $n \in [2, N-1]$. (The “extra” node is the merged source of \mathcal{G}' and target of \mathcal{G}'' .) The algorithm: (i) recursively generates AREA-maximizing schedules, Σ' for \mathcal{G}' and Σ'' for \mathcal{G}'' , in time $T(N-n+1) + T(n)$; (ii) generates Σ by concatenating Σ' and Σ'' in time $O(N)$; (iii) decomposes Σ into blocks in time $O(N)$, using the $O(N)$ blocks obtained while generating Σ' and Σ'' .

(2.b) If \mathcal{G} is a parallel composition ($\mathcal{G}' \uparrow \mathcal{G}''$) of an $(n+1)$ -node SP-DAG \mathcal{G}' and an $(N-n+1)$ -node SP-DAG \mathcal{G}'' , then, by induction, $T(N) = T(N-n+1) + T(n+1) + O(N)$, for some $n \in [2, N-2]$. (The two “extra” nodes are the merged sources and targets of \mathcal{G}' and \mathcal{G}'' .) The algorithm: (i) recursively generates AREA-maximizing schedules, Σ' for \mathcal{G}' and Σ'' for \mathcal{G}'' , in time $T(N-n+1) + T(n)$; (ii) generates Σ in time $O(N)$ using Algorithm SP-AREA; (iii) decomposes Σ into blocks in time $O(N)$, by taking the union of the blocks obtained while generating Σ' and Σ'' . (Recall that the parallel composition does not generate additional blocks.)

Overall, then, $T(N) = O(N^2)$, as claimed. □

Acknowledgment. The research of A.L. Rosenberg was supported in part by US NSF Grant CNS-0905399.

References

1. R.D. Blumofe, C.F. Joerg, B.C. Kuszmaul, C.E. Leiserson, K.H. Randall, Y. Zhou (1995): Cilk: An efficient multithreaded runtime system. *5th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PPoPP'95)*.
2. R.D. Blumofe and C.E. Leiserson (1998): Space-efficient scheduling of multithreaded computations. *SIAM J. Comput.* 27, 202–229.
3. R.D. Blumofe and C.E. Leiserson (1999): Scheduling multithreaded computations by work stealing. *J. ACM* 46, 720–748.
4. 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)*.

5. 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.
6. G. Cordasco, G. Malewicz, A.L. Rosenberg (2010): Extending IC-scheduling via the Sweep algorithm. *J. Parallel and Distributed Computing* 70, 201–211.
7. G. Cordasco and A.L. Rosenberg (2009): On scheduling DAGs to maximize area. *23rd IEEE Int. Symp. on Parallel and Distributed Processing (IPDPS'09)*, Rome, Italy.
8. G. Cordasco, A.L. Rosenberg, M. Sims (2008): Accommodating heterogeneity in IC-scheduling via task fattening. Submitted for publication. See also, On clustering tasks in IC-optimal DAGs. *37th Intl. Conf. on Parallel Processing (ICPP)*, Portland, OR, 2008.
9. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein (1999): *Introduction to Algorithms* (2nd Ed.) MIT Press, Cambridge, Mass.
10. M.R. Garey and D.S. Johnson (1979): *Computers and Intractability*. W.H. Freeman and Co., San Francisco.
11. 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.
12. A. Gerasoulis and T. Yang (1992): A comparison of clustering heuristics for scheduling DAGs on multiprocessors. *J. Parallel and Distributed Computing* 16, 276–291.
13. R. Hall, A.L. Rosenberg, A. Venkataramani (2007): A comparison of DAG-scheduling strategies for Internet-based computing. *Intl. Parallel and Distr. Processing Symp.*
14. K. Hwang and Z. Xu (1998): *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw-Hill, New York.
15. S. Jayasena and S. Ganesh (2003): Conversion of NSP DAGs to SP DAGs. MIT Course Notes 6.895.
16. D. Kondo, H. Casanova, E. Wing, F. Berman (2002): Models and scheduling mechanisms for global computing applications. *Intl. Parallel and Distr. Processing Symp.*
17. 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.
18. Y.-K. Kwok and I. Ahmad (1999): Benchmarking and comparison of the task graph scheduling algorithms. *J. Parallel and Distributed Computing* 59, 381–422.
19. Y.-K. Kwok and I. Ahmad (1999): Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys* 31, 406–471.
20. 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.
21. G. Malewicz, A.L. Rosenberg, M. Yurkewych (2006): Toward a theory for scheduling dags in Internet-based computing. *IEEE Trans. Comput.* 55, 757–768.
22. 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.
23. M. Mitchell (2004): Creating minimal vertex series parallel graphs from directed acyclic graphs. *2004 Australasian Symp. on Information Visualisation - Vol. 35*, 133–139.
24. A.L. Rosenberg (2004): On scheduling mesh-structured computations for Internet-based computing. *IEEE Trans. Comput.* 53, 1176–1186.
25. A.L. Rosenberg and M. Yurkewych (2005): Guidelines for scheduling some common computation-dags for Internet-based computing. *IEEE Trans. Comput.* 54, 428–438.
26. V. Sarkar (1989): *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, Mass.
27. J. Valdes, R.E. Tarjan, E.L. Lawler (1982): The recognition of series-parallel digraphs. *SIAM J. Comput.* 11, 289–313.