# On Constructing DAG-Schedules with Large AREAs

Scott T. Roche, Arnold L. Rosenberg, and Rajmohan Rajaraman

Northeastern University,
College of Computer and Information Science,
Boston, MA 02115, USA
{rraj,str,rsnbrg}@ccs.neu.edu

**Abstract.** The Area of a schedule $\Sigma$ for a DAG $\mathcal{G}$ is a quality metric that measures the rate at which $\Sigma$ renders $\mathcal{G}$'s nodes eligible for execution. Specifically, $AREA(\Sigma)$ is the average number of nodes of $\mathcal{G}$ that are eligible for execution as $\Sigma$ executes $\mathcal{G}$ node by node. Extensive simulations suggest that, for many distributions of processor availability and power, DAG-schedules having larger Areas execute DAGs faster on platforms that are *dynamically* heterogeneous: the platform's processors change power and availability status in unpredictable ways and at unpredictable times. (Clouds and desktop grids exemplify such platforms.) While Area-maximal schedules can provably be found for *every* DAG, efficient generators of such schedules are known only for families of well-structured DAGs. Our first result shows that the problem of crafting Area-maximal schedules for general DAGs is NP-complete, hence likely computationally intractable. The lack of efficient Area-maximizing schedulers for general DAGs has instigated the development of several heuristics for producing DAG-schedules that have large Areas. We propose a novel *polynomial-time* heuristic that produces schedules having quite large Areas; the heuristic is based on the *Sidney decomposition* of a DAG. (1) Simulations on DAGs having *random structure* yield the following results. The Sidney heuristic produces schedules whose Areas: (a) are at least $85\%$ of maximal; (b) are at least $1.25$ times greater than previously known heuristics. (2) Simulations on DAGs having the structure of *random "LEGO®"* DAGs (as formulated in earlier studies) indicate that the schedules produced by the Sidney heuristic have Areas that are at least $1.5$ times greater than previously known heuristics. The "$85\%$" result is obtained from formulating the Area-maximization problem as a Linear Program (LP); the Areas of DAG-schedules produced by the Sidney heuristic are at least $85\%$ of the Area-value produced by the (unrounded) LP. (3) The reported results on random DAGs are essentially matched by a second heuristic, which produces DAG-schedules by rounding the results of the LP formulation.

## 1 Introduction

**The problem we study.** Many modern computing platforms—notably clouds [34, 35], desktop grids [3], and volunteer-computing projects [11, 19]—exhibit extreme levels of *dynamic heterogeneity*. The availability and relative computing powers of such platforms' computing resources can change at unexpected times and in unexpected ways. Scheduling a computation for efficient execution on such a platform can be quite challenging, particularly when there are dependencies among the computation's constituent *chores*[1] (jobs, tasks, etc.); as is traditional, we model such computations as DAGs (directed acyclic graphs). The *Area* of a schedule $\Sigma$ for a DAG $\mathcal{G}$ is a quality metric that measures the rate at which schedule $\Sigma$ renders $\mathcal{G}$'s nodes eligible for execution: the larger the better. Specifically, $AREA(\Sigma)$ is the average number of nodes of $\mathcal{G}$ that are eligible for execution as $\Sigma$ executes $\mathcal{G}$ node by node. The intuition motivating the Area metric is that increasing the likelihood of having nodes eligible for execution increases the opportunities to avail oneself of available computational resources, thereby decreasing the likelihood that a computation will stall for lack of eligible work. Although this is just mathematical/computational intuition—the definition of Area does not mention any properties of the computing platform—extensive simulations ([5, 6]) suggest that, for many distributions of processor availability and power, DAG-schedules that have larger Areas execute DAGs faster on dynamically heterogeneous platforms. The current paper is motivated by the fact that, while all DAGs provably admit Area-maximizing schedules, we know how to derive such schedules *efficiently* only for a variety of specific families of DAGs [5, 8].

[1] Since a computation can be scheduled at a variety of levels of coarseness, we use the granularity-neutral term "chore" to denote the units that combine to form the computation.

**Our contributions**. The first main result of the current study establishes that the AREA-MAX *problem*, i.e., the (decision version of the) problem of generating Area-maximizing schedules for general DAGs, is NP-complete; see Section 3. It is, therefore, likely that this optimization problem is computationally intractable in general. In response, we have been seeking an *approximation algorithm* for AREA-MAX, i.e., an algorithm that produces DAG-schedules whose Area is within a fixed factor of the area of an area-maximizing schedule. We have not yet discovered a good approximation algorithm yet, but the search has not been fruitless. In Section 3, we present an algorithm that achieves AREA always within $1/(2\sqrt{n})$ of the optimum AREA. The preceding approximation factor is weak for large DAGs, so the main focus of our work has been on developing heuristics that work well for large classes of DAGs. The second main result of the current study is a new, *polynomial-time*, heuristic for producing DAG-schedules which is based on the *Sidney decomposition of a* DAG [31]. Simulation experiments suggest that the schedules produced by this Sidney heuristic have quite large areas; see Section 4.1. Specifically:

1. Simulations on DAGs having *random structure* yield the following results. The Sidney heuristic produces schedules whose Areas: (a) are at least $85\%$ of maximal; (b) are at least $1.25$ times larger than the Areas of schedules produced by previously known heuristics.
2. Simulations on DAGs having the structure of *random* "LEGO®" DAGs (so named for the toy; cf. [6]) indicate that the schedules produced by the Sidney heuristic have Areas that are at least $1.5$ times larger than the Areas of schedules produced by previously known heuristics.

The third main contribution of the current study is a new formulation of the Area-maximization problem as a Linear Program (LP, for short); see Section 4.2. The LP formulation yields two benefits. First, the Area-value produced by the (unrounded) LP for a DAG $\mathcal{G}$ affords us an upper bound on the maximal Area achievable by any schedule for $\mathcal{G}$. Indeed, this bound gives us access to the "$85\%$" result just mentioned for the Sidney heuristic. Second, the LP formulation yields a second new polynomial-time heuristic for the Area-maximization problem. While the LP heuristic just essentially matches the random-DAG Areas achieved by the Sidney heuristic's schedules, the LP schedules promise to yield valuable information about the structure of Area-maximizing schedules, in the manner discussed in Section 4.2.

**Related Work**. The problem of scheduling a computation on a parallel/distributed computing platform, with the goal of minimizing job completion time, has been studied since the development of such platforms [30]. Most variants of this problem provide nontrivial computational challenges, esecially when the constituent chores of the computation of interest have inter-chore dependencies that constrain the order of chores' executions. In common with most of the scheduling literature, we represents a job and its inter-chore dependencies as a directed acyclic graph (DAG) each of whose arcs exposes one chore that cannot be executed before some other chore. An extensive overview of DAG-scheduling algorithms related to grids is given in [10]. Most significant scheduling problems on DAGs are, even under simplified assumptions, computationally intractable, which has led to the development of a multitude of heuristics; cf. [20].

Despite differences in detail, virtually all proposed strategies for scheduling DAGs rely on knowing, possibly in a stochastic sense as in [36], (almost) exact execution times of chores; for this reason, dynamically heterogeneous platforms resist all standard scheduling strategies. To address this situation, a number of attempts have been made to adapt earlier DAG-scheduling heuristics, such as HEFT [32] and FCP [28], to the new platforms. None of these attempts have successfully addressed the range of challenges posed by dynamically heterogeneous platforms.

Among the bold approaches to crafting DAG-schedules for dynamically heterogeneous platforms are the partial-order schedules of [27], which strive to craft schedules that enjoy temporal flexibility that is solidified only at run time. A similar delay-of-commitment strategy forms the response advocated in [2, 17] to the unpredictability of highly volatile computing platforms. Yet other studies propose scheduling strategies wherein a precomputed static schedule is reorganized at run time in response to changes in processors' powers [36, 23]; one instance of this appears in [26], where planned checkpoints allow one to react dynamically to unexpected behavior by a volatile platform.

The scheduling strategy that leads to the current study was initiated in [29, 25]. These sources advocated ignoring the (unknowable) characteristics of the host platform and, instead, deploying the chores of a DAG in an order that maximized the rate of producing more chores that are eligible for deployment. The intuition is that under this regimen, which is called *IC-scheduling*, whenever processors speed up, one will be more likely to have work to allocate to them. Simulation experiments in [16, 24] seem to validate this intuition, but an unrecoverable flaw in IC-scheduling was discovered in [25]—many DAGs do not admit optimal schedules under this paradigm (although many computationally significant DAGs do admit such schedules [7]). This discovery led to the development of *Area-maximizing*

DAG-scheduling, whose study we continue here. The fundamentals of Area-maximizing scheduling are established in [5] where, among other results, it is shown that $(a)$ every DAG admits an optimal Area-maximizing schedule and $(b)$ optimal Area-maximizing schedules and optimal IC-schedules coincide for any DAG that admits an optimal IC-schedule. Since *efficient* generators are not known for Area-maximizing schedules, a heuristic was developed in [6] that converted a DAG $\mathcal{G}$ to a *series-parallel* version $\sigma(\mathcal{G})$ and then generated a schedule for $\mathcal{G}$ by "filtering" an Area-maximizing schedule for $\sigma(\mathcal{G})$ (obtained via the algorithm in [8]). The simulation experiments reported in [6] suggest that Area-maximizing DAG-scheduling has computational benefits similar to those of IC-scheduling, although to a moderated degree. The current study focuses on a new heuristic whose schedules have Areas larger than those of the schedules of [6].

There have also been studies that focus on DAG-scheduling strategies rather than complete schedules. An interesting comparison of two dynamic approaches appears in [18]: replicated allocation of chores vs. deadline-triggered reallocation. Other sources have analyzed the reliability of scheduling DAGs under execution-time uncertainty [14, 22]. Finally, one finds in [1] a framework for minimizing makespan when processors proceed asynchronously, executing DAGs having unit-time chores.

Throughout, we exploit a nonobvious connection between the AREA-MAX problem and the *Minimum Weighted-Completion-Time* problem for DAGs, MWCT. In Section 3, we invoke a result from [33] to establish the NP-Completeness of our problem; in Section 4.1, we draw inspiration from [21] to develop a new heuristic for producing large-Area schedules for DAGs.

## 2 Computation-DAGs and Their Schedules

**Basic Definitions**. Computation-DAGs. A *(computation-)*DAG $\mathcal{G}$ has a set $\mathcal{N}_{\mathcal{G}}$ of cardinality $N_{\mathcal{G}}$ comprising its *nodes*, each representing a chore in a computation, and a set $\mathcal{A}_{\mathcal{G}}$ of cardinality $A_{\mathcal{G}}$ comprising its *arcs*, each representing an intertask dependency. For arc $(u \to v) \in \mathcal{A}_{\mathcal{G}}$: • chore $v$ cannot be executed until chore $u$ is; • $u$ is a *parent* of $v$; $v$ is a *child* of $u$ in $\mathcal{G}$. The notion of the *ancestor* of a node is inherited from the notion of parent. The *indegree* (resp., *outdegree*) of $u \in \mathcal{N}_{\mathcal{G}}$ is its number of parents (resp., children). A parentless node is a *source*; a childless node is a *sink*. $\mathcal{G}$ is *bipartite* if $\mathcal{N}_{\mathcal{G}}$ can be partitioned into $X$ and $Y$, and each arc $(u \to v)$ has $u \in X$ and $v \in Y$; we say that $\mathcal{G}$ is bipartite *of type* $(X \to Y)$.

DAG schedules and their quality. When one executes a DAG $\mathcal{G}$, a node $v \in \mathcal{N}_{\mathcal{G}}$ becomes eligible (for execution) only after all of its parents have been executed. (Hence, sources are always eligible.) We do not allow recomputation of nodes, so a node loses its eligible status once it is executed. In compensation, the execution of a node $v \in \mathcal{N}_{\mathcal{G}}$ may render new nodes eligible; this occurs when $v$ is their last-executed parent. A *schedule* $\Sigma$ for a DAG $\mathcal{G}$ is a rule for selecting which eligible node to execute at each step of an execution of $\mathcal{G}$. $\Sigma$ is, thus, a *topological sort* [9] of $\mathcal{G}$, i.e., a linearization $\Lambda_{\Sigma}$ of $\mathcal{N}_{\mathcal{G}}$ in which all children of each node $v$ appear after $v$.

We measure the quality of a schedule $\Sigma$ via the rate at which $\Sigma$'s successive node-executions produce new eligible nodes—the more, the better. Because many DAGs do not admit schedules that execute nodes so that the number of eligible nodes on $\mathcal{G}$ is maximized *at every step of the computation*—these are the *IC-optimal* schedules of [25]—our goal is schedules that maximize the *average* number of eligible nodes on $\mathcal{G}$, averaged over all steps of the computation. A schedule that achieves this goal is said to be *AREA-maximizing*, as explained in the next subsection.

**AREA-maximizing schedules**. The AREA metric. For any schedule $\Sigma$ for a DAG $\mathcal{G}$ and any integer $T \in [0, N_{\mathcal{G}}]$,[2] we denote by $E_{\Sigma}(T)$ the number of nodes of $\mathcal{G}$ that are eligible at step $T$ when $\Sigma$ executes $\mathcal{G}$.[3] The *eligibility profile* of schedule $\Sigma$ is the $(N_{\mathcal{G}}+1)$-tuple $\Pi(\Sigma) = \langle E_{\Sigma}(0), E_{\Sigma}(1), \ldots, E_{\Sigma}(N_{\mathcal{G}})\rangle$. The *AREA* of $\Sigma$ is the sum

$$AREA(\Sigma) = E_{\Sigma}(0) + E_{\Sigma}(1) + \cdots + E_{\Sigma}(N_{\mathcal{G}}). \tag{1}$$

Note that $AREA(\Sigma)$ is the unnormalized average number of nodes of $\mathcal{G}$ that are eligible when $\Sigma$ executes $\mathcal{G}$.[4] Our goal is to find, for each DAG $\mathcal{G}$ an *AREA-maximizing schedule* (*A-M schedule*, for short), i.e., a schedule $\Sigma^{\star}$ for $\mathcal{G}$ such

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

[3] We measure time in an event-driven manner, as the number of nodes executed to that point.

[4] The term *Area* arises by analogy with the approximation of integrals by Riemann sums.

that

$$AREA(\Sigma^\star) \;=\; \max_{\Sigma \text{ a schedule for } \mathcal{G}} AREA(\Sigma) \;\stackrel{\text{def}}{=}\; AREA(\mathcal{G}).$$

We refer to the quest for A-M schedules as the AREA-MAX problem.

Streamlining the metric. The following lemma technically simplifies AREA-MAX.

**Lemma 1 ([5]).** *Every* DAG $\mathcal{G}$ *admits an A-M schedule* $\Sigma$ *that executes* $\mathcal{G}$*'s sinks only after executing all of its nonsinks.*

Focus on a DAG $\mathcal{G}$ that has $n$ nonsinks, $N$ nonsources, $s$ sources, and $S$ sinks (so that $N_{\mathcal{G}} = s + N = S + n$).

**1**. If a schedule $\Sigma$ for $\mathcal{G}$ honors Lemma 1, then the last $S$ entries in $\Pi(\Sigma)$ are: $S-1, \ldots, 1, 0$. We can, therefore, maximize $AREA(\Sigma)$ by maximizing

$$Area(\Sigma) \;\stackrel{\text{def}}{=}\; \sum_{i=0}^{n} E_\Sigma(i) \;=\; AREA(\Sigma) \;-\; \binom{S}{2}. \tag{2}$$

**2**. Let $e_\Sigma(t)$ denote the number of nodes of $\mathcal{G}$ that are rendered eligible by the node-execution at step $t \in [1, N_{\mathcal{G}}]$ of $\Sigma$. By (2), $E_\Sigma(t) = s - t + \sum_{j=1}^{t} e_\Sigma(j)$, so that

$$Area(\Sigma) \;=\; \sum_{t=0}^{n}\sum_{j=1}^{t} e_\Sigma(j) \;+\; (n+1)s \;-\; \binom{n+1}{2},$$

which exposes

$$area(\Sigma) \;\stackrel{\text{def}}{=}\; \sum_{t=0}^{n}\sum_{j=1}^{t} e_\Sigma(j) = n \cdot e_\Sigma(1) + (n-1) \cdot e_\Sigma(2) + \cdots + 1 \cdot e_\Sigma(n) \tag{3}$$

as the only portion of $Area(\Sigma)$ that actually depends on choices made by $\Sigma$.

## 3 The NP-Completeness of AREA Maximization and a $\sqrt{n}$-Approximation Algorithm

It is clear that the (decision version of the) AREA-MAX Problem lies within the class NP. Given a DAG $\mathcal{G}$ and integer $k$, one can "guess" a topological sort $\mathcal{T}$ of $\mathcal{G}$ and determine whether $AREA(\Sigma) \leq k$ for the schedule $\Sigma$ embodied in $\mathcal{T}$. We show now that AREA-MAX is also NP-hard, so that the problem is NP-complete. Our proof is via reduction from the 0-1 *Minimum Weighted-Completion-Time* problem for a class of bipartite DAGs. This problem, which we refer to as $(0,1)$-MWCT, is defined as follows.

One is given a *bipartite* DAG $\mathcal{G}$, with one set of the partition consisting of source nodes, and the other set consisting of sink nodes. Thus, $\mathcal{G}$ is a bipartite DAG of type $(\mathcal{S} \to \mathcal{T})$, where $\mathcal{S}$ is a set of $s$ sources and $\mathcal{T}$ is a set of $S$ sinks. Every source $u \in \mathcal{S}$ has computation time $C_u = 1$ and weight $w_u = 0$, while every sink $v \in \mathcal{T}$ has computation time $C_v = 0$ and weight $w_v = 1$. Under this model, the makespan when we execute $\mathcal{G}$ is not affected by when we execute any sink $v$—as long, of course, as $v$ is eligible. For definiteness, we will execute each sink "greedily," i.e., as soon as it becomes eligible, and we will henceforth view a schedule as an ordering of $\mathcal{G}$'s $s$ sources. The *weighted completion time* for $\mathcal{G}$ associated with schedule $\Sigma$ is

$$W_\Sigma \;\stackrel{\text{def}}{=}\; \sum_{j \in \mathcal{N}_{\mathcal{G}}} w_j C_j \;=\; 1 \cdot e_\Sigma(1) + 2 \cdot e_\Sigma(2) + \cdots + s \cdot e_\Sigma(s). \tag{4}$$

Our challenge is to find a schedule $\Sigma$ for $\mathcal{G}$ that has minimal $W_\Sigma$. It is shown in [33] that $(0,1)$-MWCT is NP-Complete.

Employing standard job-scheduling notation, we refer to the preceding problem as a $1|prec|\sum w_j C_j$ problem, meaning that it involves a computation by a single processor (1), permits inter-job precedences ($prec$), and strives to minimize the expression $\sum w_j C_j$.

The reduction from $(0,1)$-MWCT to AREA-MAX resides in the following lemma.

**Lemma 2.** *Any AREA-maximizing schedule for the* 0-1 *bipartite* DAG $\mathcal{G}$ *solves the* $1|prec|\sum w_jC_j$ *problem for* $\mathcal{G}$*: it minimizes* $\mathcal{G}$*'s weighted completion time.*

*Proof.* Let $\Sigma$ be an arbitrary schedule for the 0-1 bipartite DAG $\mathcal{G}$. For definiteness, let $\Sigma$ execute $\mathcal{G}$'s $s$ sources in the order $u_1, \ldots, u_s$. Then, we have

$$W_\Sigma = \sum_{k=1}^{s} k e_\Sigma(k). \tag{5}$$

Adding Eqs. (3) and (5), we obtain

$$Area(\Sigma) + W_\Sigma \;=\; (S+1)s - \binom{s+1}{2} + (s+1)\sum_{k=1}^{s} e_\Sigma(k) \;=\; (S+1)s - \binom{s+1}{2} + (s+1)S,$$

since each of the $S$ sinks becomes eligible exactly once. It thus follows that any schedule that any AREA-maximizing schedule also minimizes the weighted completion time, whence the result.

**A $\sqrt{n}$-approximation algorithm for AREA maximization.** Given the NP-completeness of the problem, it is unlikely that AREA maximization is computationally tractable in general. We have been seeking provable approximation algorithms for the problem. Since the AREA of every schedule is at least $n$ and at most $n(n+1)/2$, we have a trivial $(n+1)/2$-approximation to the problem. We have been exploring different approaches with the hope of achieving a much better approximation, say $O(1)$ or polylogarithmic factor in the number of jobs.

The best approximation factor we are able to show is an $2\sqrt{n}$-approximation, which we now present.

1. Using a maximum flow algorithm, find the largest antichain of the DAG $\mathcal{G}$; i.e., a maximum-size set $S$ of nodes such that there is no path in the dag between any pair of nodes in $S$.
2. Decompose $\mathcal{G}$ into three DAGs: (i) $\mathcal{G}_1$ is the DAG induced by all nodes that have a nonempty path to some node in $S$; (ii) $\mathcal{G}_2$ is the DAG induced by $S$ (note it has no edges); (iii) and $\mathcal{G}_2$ is the DAG induced by all nodes that have no path to any node in $S$.
3. Let $\Sigma_1$, $\Sigma_2$, and $\Sigma_3$ denote arbitrary topological sorted orders of $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$, respectively. Return the schedule $\Sigma$, which is the concatenation of $\Sigma_1$, $\Sigma_2$, and $\Sigma_3$ in order.

**Lemma 3.** *The above algorithm computes a schedule with area at least* $1/(2\sqrt{n})$ *of the optimal.*

*Proof.* Let $w$ be the size of the largest antichain of $\mathcal{G}$ (also referred to as width of $\mathcal{G}$). Since the set of eligible jobs at any instant of any schedule is an antichain, the AREA of any schedule for $\mathcal{G}$ is at most $nw$. We consider two cases depending on the value of $w$. If $w \leq \sqrt{n}$, then since the AREA of any schedule is at least $n$, it is at least $1/sqrtn$ of the maximum area. If $w > \sqrt{n}$, then we have:

$$\begin{aligned} AREA(\Sigma) &= AREA(\Sigma_1) + AREA(\Sigma_2) + AREA(\Sigma_3) \\ &\geq n - w + AREA(\Sigma_2) \\ &= n - w + w(w+1)/2. \end{aligned}$$

So the ratio of $AREA(\Sigma)$ and the optimum is at least

$$\frac{n + w^2/2 - w/2}{nw} \geq \frac{w^2}{2nw} \geq \frac{1}{2\sqrt{n}}.$$

We have thus shown that the above algorithm yields a $2\sqrt{n}$-approximation to the AREA maximization problem.

## 4 Two New Heuristics

### 4.1 A DAG-Scheduling Heuristic Based on the Sidney Decomposition

This section is devoted to developing the SIDNEY scheduling heuristic. The heuristic builds on a transformation of the input DAG $\mathcal{G}$ into a modified DAG $\mathcal{G}'$. We show that finding an AREA-maximizing schedule for $\mathcal{G}$ is *equivalent* to finding a schedule that minimizes the weighted completion time $\sum_{v \in \mathcal{N}_{\mathcal{G}'}} w_v C_v$ for $\mathcal{G}'$. Here, we reduce the AREA-MAX problem to the Minimum Weighted Completion Time problem, and use a known approximation algorithm for the latter problem to derive a heuristic for area maximization. (See Section 3 for notation and terminology.)

Given a DAG $\mathcal{G}$, we construct its 0-1 *version* $\mathcal{G}_{0,1}$ as follows. The nodes of $\mathcal{G}_{0,1}$ are obtained by splitting every node $v \in \mathcal{N}_{\mathcal{G}}$ into two nodes, $v_0$ and $v_1$. Each node of $\mathcal{G}_{0,1}$ with a 0 subscript (we call these the **zero-nodes**) has a processing time of 0 and a weight of 1: $p_{v_0} = 0$ and $w_{v_0} = 1$; each node of $\mathcal{G}_{0,1}$ with a 1 subscript (the **one-nodes**) has a processing time of 1 and a weight of 0: $w_{v_1} = 0$ and $p_{v_1} = 1$. Finally, we give $\mathcal{G}_{0,1}$ an arc $(u_1 \to v_0)$ for each arc $(u \to v)$ of $\mathcal{G}$ and an arc $(u_0 \to u_1)$ for each node $u$ of $\mathcal{G}$.

Let $\Sigma$ be a schedule for $\mathcal{G}$, and let $\Sigma'$ be a schedule for $\mathcal{G}_{0,1}$. We call $\Sigma'$ a 0-1 *version* of $\Sigma$ if its ordering of $\mathcal{G}_{0,1}$'s one-nodes is consistent with $\Sigma$'s ordering of $\mathcal{G}$'s nodes. Thus if $\Sigma$ executes $\mathcal{G}$'s nodes in the order $v_1, v_2, \ldots, v_{N_{\mathcal{G}}}$, then $\Sigma'$ executes $\mathcal{G}_{0,1}$'s nodes in the order $v_{1,1}, v_{2,1}, \ldots, v_{N_{\mathcal{G}},1}$.

**Lemma 4.** *Let $\mathcal{G}$ be any* DAG *and let $\mathcal{G}_{0,1}$ be its* 0-1 *version. If a schedule $\Sigma$ is AREA-maximizing for $\mathcal{G}$, then any* 0-1 *version of $\Sigma$ minimizes weighted completion time for $\mathcal{G}_{0,1}$.*

*Proof.* Because every zero-node has processing time 0, any WCT schedule for $\mathcal{G}_{0,1}$ will execute each zero-node as soon as it is eligible. Therefore, the WCT time for any 0-1 version of schedule $\Sigma$ is given by

$$W_{\Sigma'} \;=\; \sum_{i \in \mathcal{N}_{\mathcal{G}}} w_i \cdot \max_{j \text{ a parent of } i} C(j) \;=\; \sum_{i \in \mathcal{N}_{\mathcal{G}}} w_i \cdot E(i), \tag{6}$$

where $C(j)$ denotes the time when node $j$'s execution completes, and $E(i)$ denotes the time when node $i$ becomes eligible. By similar reasoning, we can represent $AREA(\Sigma)$ as follows.

$$AREA(\Sigma) \;=\; \sum_{i \in \mathcal{N}_{\mathcal{G}}} \big((C(i) - 1) - E(i)\big) \;=\; \binom{n+1}{2} - \sum_{i \in \mathcal{N}_{\mathcal{G}}} E(i). \tag{7}$$

It follows from equations (6) and (7) that if one could replace $\Sigma'$ by a schedule $\Sigma''$ for $\mathcal{G}_{0,1}$ such that $W_{\Sigma''} < W_{\Sigma'}$, then there would exist $\widehat{\Sigma}$ for $\mathcal{G}$ such that $AREA(\widehat{\Sigma}) > AREA(\Sigma)$. But schedule $\widehat{\Sigma}$ would contradict $\Sigma$'s assumed AREA-maximality. The lemma follows.

The duality between $AREA(\mathcal{G})$ and $W_{\mathcal{G}_{0,1}}$ for any DAG $\mathcal{G}$ enables us to invoke a known approximation algorithm for minimum weighted completion time to $\mathcal{G}_{0,1}$ to obtain a heuristic algorithm for AREA maximization. We proceed by invoking an algorithm introduced by Sidney [31] and later reused in [4]. We provide only an overview of the process here, referring the reader to the cited papers for details.

In Sidney's work, each node of a DAG $\mathcal{G}$ to be decomposed using Sydney's algorithm, each node/task $i$ has a *processing time* $p_i$ and a *weight* $w_i$. The *rank* of node $i$ is the quotient $r_i = p_i/w_i$; the rank of a set $S$ of nodes is $r(S) = \sum_{i \in S} p_i/w_i$. Following [4], we say that a sub-DAG $\mathcal{G}'$ of $\mathcal{G}$ is *precedence-closed* if for each node $i$ of $\mathcal{G}'$, every ancestor of $i$ is also in $\mathcal{G}'$. Additionally, $\mathcal{G}*$ denotes a precedence-closed subgraph of $\mathcal{G}$ of minimum rank. Finally, a *segment* in a schedule $\Sigma$ is any set of nodes that are scheduled consecutively by $\Sigma$. It is proved in [31]—and was rediscovered in [4]—that there is a generalization of Smith's rule for precedence-constrained graphs:

**Lemma 5.** *There exists an optimal schedule for* DAG *$\mathcal{G}$ in which an optimal schedule for $\mathcal{G}*$ occurs as a segment that starts at time zero.*

In other words, in an optimal schedule $\Sigma$ for $\mathcal{G}$, an optimal ordering of the nodes of the minmum-rank precedence-closed subgraph $\mathcal{G}*$ appears as the first segment of the schedule. The goal in [4] is to exhibit a polynomial-time algorithm **A** that (1) recursively finds the set of nodes of $\mathcal{G}*$ (on the residual graph $\mathcal{G} - \mathcal{G}*$), and then (2) schedules the

nodes within each set in any feasible schedule, such that **A** finds a schedule $\Sigma$ for $\mathcal{G}$ whose weighted completion time is at most double the optimal minimum weighted completion time for $\mathcal{G}$. The polynomial-time algorithm **A** works with the Sidney decomposition of $\mathcal{G}$ in a manner we outline here.

Rather than specifically looking for the minimum-rank precedence-closed sub-DAG $\mathcal{G}*$, algorithm **A** finds a sub-DAG whose rank is at most a constant $\lambda > 0$ specified as an input to the algorithm. It accomplishes this by constructing an associated *capacitated* graph $\mathcal{G}_\lambda$, with the following properties:

- The nodes of $\mathcal{G}_\lambda$ consist of the nodes of $\mathcal{G}$ along with a source $s$ and a sink $t$.
- The arcs of $\mathcal{G}_\lambda$ are: $\{(s \to t), (i \to t) | i \in \mathcal{N}(\mathcal{G})\} \bigcup \{(i \to j) | j \text{ is an ancestor of } i\}$.

We further associate a capacity $c(e)$ for every arc, as follows:

$$c(e) \;=\; \begin{cases} p_i, & \text{if } e = e(i,t) \\ \lambda w_i, & \text{if } e = (s,i) \\ \infty, & \text{otherwise} \end{cases}$$

Finding a subgraph $\mathcal{G}*$ is thus reduced to finding a $(s,t)$-minimum cut for $\mathcal{G}_\lambda$ with cut value at most $\lambda w(\mathcal{G})$. Lemma 3 in [4] guarantees that if $(A, B)$ is such a cut, then the rank of $A - \{s\}$ is less than $\lambda$, and $A - \{s\}$ is precedence-closed in $\mathcal{G}$. Hence, to find $\mathcal{G}*$, one can perform a binary search on $\lambda$ to find $\mathcal{G}*$, and then recurse on the residual DAG $\mathcal{G} - \mathcal{G}*$ until the entire DAG has been decomposed. Alternatively, one can use an algorithm such as that described in [12] to find all points $\lambda$ in a single max-flow computation (making use of a variable of the push-relabel algorithm), thereby decomposing the entire DAG $\mathcal{G}$ in a single pass. This alternative provides an efficient running time of $O(\min(n^{2/3}, m^{1/2}) m \log(n^2/m) \log U)$, where $n$ is the number of nodes in the DAG, $m$ is the number of arcs, and $U$ the maximum (finite) capacity, which is at most $n$ in our case. Thus, for sparse dags ($m = O(n)$), the running time is $O(n^{5/3} \log^2 n)$, while for dense dags ($m = \Theta(n^2)$), the running time is $O(n^{5/2} \log^2 n)$.

As the final ingredient for creating the desired SIDNEY heuristic, we recall the DYNAMIC-GREEDY scheduling heuristic from [5].

The DYNAMIC-GREEDY heuristic schedules a DAG $\mathcal{G}$ by maintaining a MAX-priority queue of the eligible nodes, (partially) ordered by their *yields*.

> The *yield* of an eligible node $v \in \mathcal{N}_\mathcal{G}$ at step $t$ of a schedule's execution of $\mathcal{G}$ is the number of nodes that would be rendered eligible if the schedule were to execute $v$ at that step.

At each step, a maximal-yield node is selected for execution. When a node completes executing, all newly eligible nodes are inserted into the priority queue, in random order. (The heuristic thus makes an optimal choice for this step, but it ignores future ramifications of this choice.)

Finally, we are ready to specify the SIDNEY heuristic for computing a large-AREA DAG-schedule:

**The SIDNEY heuristic**

Given a DAG $\mathcal{G}$:

1. Construct the associated 0-1 DAG $\mathcal{G}_{0,1}$.
2. Use a max-flow computation to perform a Sidney decomposition of $\mathcal{G}_{0,1}$.
3. Let $S_1, \ldots, S_k$ be the node-sets computed in the Sidney decomposition:
   (a) Remove all 0-nodes from each task-set $S_i$.
   (b) For each task-set $S_i$, use the DYNAMIC-GREEDY heuristic to produce a schedule $\Sigma_i$ for the nodes in $S_i$.
4. Output schedule $\Sigma \;=\; \Sigma_1 \Sigma_2 \ldots \Sigma_k$, the concatenation of the $k$ subschedules.

### 4.2 A DAG-Scheduling Heuristic Based on Linear Programming

Because the AREA-MAX Problem involves maximizing a sum of rather simple terms, it is not surpising that it can be formulated as a Linear Program (LP). We present such a formulation that serves two purposes in our study. First, the (unrounded) solution produced by the LP for any DAG $\mathcal{G}$ provides an upper bound on the maximal Area of $\mathcal{G}$ under any schedule. Second, if one rounds the solution to yield *integer* values, one obtains a valid schedule for $\mathcal{G}$. This (solve

LP)-(then round) procedure comprises the *LP heuristic* for scheduling DAG $\mathcal{G}$. Of course, the problem of obtaining *optimal* integer solutions from an LP is the well-known *ILP problem*, which is NP-hard in general, hence likely computationally intractable [13]. However, there may be ways to simplify the ILPs that arise in AREA-maximization, at least for large classes of DAGs, so as to either simplify the problem computationally or to have access to approximate solutions via the unrestricted (non-integer) form of the Linear Program. Even lacking such simplification, the solution to the LP formulation for DAG $\mathcal{G}$ provides us with an upper bound on the Area achievable via any DAG schedule.

Let DAG $\mathcal{G}$ have $n$ nodes. Our LP-formulation of the AREA-MAX problem employs three classes of indicator (i.e., 0-1 valued) variables, each of size roughly $n^2$. For $i \in [1, n]$ and $t \in [0, n]$:

| Variable | Interpretation |
|---|---|
| $x_{i,t}$ | *Task/node $i$ is executed at step $t$ of schedule $\Sigma$.* |
| $y_{i,t}$ | *Task/node $i$ is eligible at step $t$ of schedule $\Sigma$.* |
| $z_{i,t}$ | *Task/node $i$ has been executed prior to step $t$ of schedule $\Sigma$.* |

Since any schedule for $\mathcal{G}$ executes one eligible node per step, the AREA-MAX problem can now be formulated as follows; cf. (2).

$$\text{maximize } \sum_{t=0}^{n} \sum_{i=0}^{n} y_{i,t} \text{ subject to:} \tag{8}$$

$$\sum_{t=0}^{n} x_{i,t} = 1 \text{ for all } i \in \mathcal{N}_{\mathcal{G}} \tag{9}$$

$$\sum_{i=0}^{n} x_{i,t} = 1 \text{ for all } t \in [0, n] \tag{10}$$

$$z_{i,T} = \sum_{t<T} x_{i,t} \text{ for all } i \in \mathcal{N}_{\mathcal{G}} \text{ and } T \in [1, n] \tag{11}$$

$$y_{i,T} = 1 - z_{i,T} \text{ for all } i \in (\text{Sources of } \mathcal{G}) \text{ and } T \in [1, n] \tag{12}$$

$$y_{i,T} \leq z_{j,T} - z_{i,T} \text{ for all } T \in [0, n] \text{ and } j \in (\text{ancestors of } i) \tag{13}$$

$$z_{i,T+1} \leq z_{j,T} \text{ for all } T \in [0, n] \text{ and } j \in (\text{ancestors of } i) \tag{14}$$

$$x_{i,t}, y_{i,t}, z_{i,t} \in \{0, 1\} \text{ for all } i \in \mathcal{N}_{\mathcal{G}} \text{ and } t \in [0, n] \tag{15}$$

Constraints (9,10) ensure, respectively, that each node/task is completed and that no processor is idle at any time step. Constraint (11) ensures that the cumulative-execution variable $z$ equals the sum of work done on each node prior to time $T$. Constraint (12) ensures that the eligibility of a source node is 1 minus (the work already completed in prior time steps). Constraint (13) ensures that for each precedence constraint ($j$ must be executed before $i$), the eligibility of a node is bounded by the work already done on it minus the work already done on all of its ancestors. Finally, constraint (14) ensures that for each precedence constraint ($j$ must be executed before $i$), the work done on node $i$ is no greater than the work done on all of its ancestors.

To create a Linear Program (LP) from this ILP, we can replace the integrality constraints 15 with the following:

$$0 \leq y_{i,t} \leq 1 \text{ for all } i, t \tag{16}$$

$$0 \leq x_{i,t} \leq 1 \text{ for all } i, t \tag{17}$$

This transition from the ILP formulation to the LP formulation preserves almost all of the qualities of the ILP version of AREA-MAX. The major difference is that the LP formulation allows fractional execution (and eligibility) of tasks—which is equivalent to allowing preemption in schedules. The LP thus provides an upper bound for the optimal value of $AREA(\mathcal{G})$.

The preceding development leads us naturally to the following DAG-scheduling heuristic.

**The LP heuristic**

Given a DAG $\mathcal{G}$:

1. Construct the LP from $\mathcal{G}$.
2. Solve the LP
3. For each node $i \in \mathcal{N}_{fg}$, calculate a completion time $C_i$ as the first time step $T$ such that $\sum_{t=0}^{T} x_{i,t} = 1$
4. Sort the list of completion times $\{C_i\}_{i=0}^{n}$, breaking ties arbitrarily. The resulting ordering of $\mathcal{N}_{fg}$ is schedule $\Sigma$.
5. Output $\Sigma$ as the schedule for $\mathcal{G}$.

# 5 Simulation Experiments

## 5.1 Experimental Procedure

**Overview of the experiment.** To test the AREA quality of our SIDNEY and LP heuristics, we generated synthetic DAGs that share structural characteristics with a variety of "real" computation-DAGs, especially those encountered in scientific computing. For most experiments, we constructed schedules for each DAG using three heuristics: the SIDNEY heuristic of Section 4.1 and the two "best" known heuristics (described below) as determined by the experiments described in [6]. We then compared, for each DAG, the AREA of the three generated schedules. For some small DAGs (having 100 nodes or fewer), we were able to consider also the AREAs of the schedules produced by the LP heuristic of Section 4.2. While LP is a polynomial-time heuristic, its current implementation is prohibitively computationally intense, certainly moreso than the SIDNEY heuristic. We are continuing to seek avenues to accelerate the LP computation since its specification for a DAG $\mathcal{G}$ exactly reflects the definition of $AREA(\mathcal{G})$.

**The DAGs we tested.** We generated random DAGs from the following families for our experiment.

1. *Random $n$-node* DAG*s*. We ordered $n$ nodes for our DAG into a random sequence $1, 2, \ldots, n$. We designates the last five nodes in the sequence as sinks and then, for each node $i \in \{1, \ldots, n-5\}$, we randomly selected five children, $j_1 > i, \ldots, j_5 > i$ and generated arcs $(i \to j_k)$.
2. *Random $n$-node LEGO®-*DAG*s*. We tested LEGO®-DAGs (so named for the toy), as defined in [25]. These DAGs are built from a repertoire of *Bipartite Building Block* DAG*s (BBBs)*, that represent (parallel) steps in a computation. As in [25] (q.v.), we employed BBBs that reflect a single: *expansive step* (as in an out-tree), *reductive step* (as in an in-tree), *group step* (as encountered in computations exemplified by convolutions or parallel-prefix operations). We selected BBBs, randomized according to both size and structure, and composes them to create multi-step, multi-level computations; we continued selecting and composing BBBs until the resulting LEGO®-DAG reached the desired size range. We created two classes of LEGO®-DAGs, one using BBBs whose sizes are drawn from a uniform distribution in the range $[2, 20]$ and another using BBBs whose sizes are drawn from a harmonic distribution that produces building blocks of expected size 10.

**The heuristics we tested.** The three schedulers we used to generate schedules were

1. The SIDNEY HEURISTIC scheduler, whose structure is described in Section 4.1.
2. The AOSPD scheduler. In brief, this scheduler which is developed in [8], takes the input DAG $\mathcal{G}$ and (*if $\mathcal{G}$ is not already a series-parallel* DAG) invokes the algorithm found in [15] to convert $\mathcal{G}$ to a *series-parallel* DAG $\sigma(\mathcal{G})$ (while retaining much of $\mathcal{G}$'s parallel structure). The AOSPD scheduler then generates an AREA-maximizing schedule for $\sigma(\mathcal{G})$, using the algorithm developed in [8]. We have chosen the AOSPD scheduler as a competitor for the SIDNEY scheduler because AOSPD has experimentally been shown to generate larger-AREA schedules than all tested heuristics that are *oblivious*, in the sense that they do not exploit any characteristics of the platform on which $\mathcal{G}$ is to be executed.
3. The DYNAMIC GREEDY scheduler, whose structure is described in Section 4.1. We include this heuristic in the competiton because it achieves the second-best AREA performance (after AOSPD) in the experiments in [6, 8].

**Experimental methodology.** DAG *sizes.* For the DAGs having random structure, we generates DAGs of sizes $n = \{100, 200, 300, 500\}$. For the LEGO®-DAGs, we generated DAGs of approximate sizes $n = 200k$ where $k \in [1, 20]$. We generated 100 DAGs of each size for each of: random DAGs, uniform-LEGO®-DAGs, and harmonic-LEGO®-DAGs. For each generated DAG, we constructed a schedule associated using each of the three heuristics and computed the resulting AREA.

### 5.2 Experimental Results

**The SIDNEY heuristic.** The plots in Fig. 1 illustrate that, for the range of DAG-classes and -sizes tested, the Areas of the schedules generated by the SIDNEY heuristic far exceed those of the schedules generated by both the previous "champion" AOSPD heuristic of [8, 6] and the best "one-step-optimal" DYNAMIC GREEDY heuristic of [5].

1. The advantage of the SIDNEY heuristic is particularly remarkable when executing both classes of LEGO®-DAGs (constructed using, respectively, a uniform and a harmonic distribution of BBBs). For these DAGs, the Areas of the SIDNEY heuristic's schedules have an advantage that is a factor of 2.3 over the schedules produced by DYNAMIC-GREEDY and a factor of nearly 1.5 over the schedules produced by the AOSPD heuristic. These numbers are particularly heartening because of the structural similarity of LEGO®-DAGs with DAGs encountered in real applications.
2. The SIDNEY heuristic exhibited a notable Area-advantage over the competing heuristics when executing random DAGs also, albeit with a somewhat smaller amplitude: roughly a factor of 1.3. We do not yet know how to interpret this decreased advantage, but it is conceivably related to the fact that random DAGs often exhibit pathological structure (by, e.g., having high expansion). We note with interest that the AOSPD heuristic's schedules exhibit a negligible Area-advantage over those of the DYNAMIC-GREEDY heuristic when executing random DAGs.

The preceding evidence suggests that the SIDNEY heuristic can find large-Area schedules for a broader class of DAGs than the scheduling policies studied in [8, 6].

**The LP heuristic.** Our experience, thus far, with the LP heuristic has led us to classify it as an auxiliary scheduler rather than a primary one (such as the SIDNEY and AOSPD heuristics). We justify this assessment.

**The cons.** We have yet to find an LP solver that can efficiently handle DAGs of even moderate size, because the associated LPs are enormous. For example, DAGs of size $n = 400$ yield LPs with $\approx 5 \times 10^5$ variables and more than $\approx 10^6$ (mostly non-sparse) constraints. Due to this computational density, we were only able to generate multiple schedules for only small random DAGs, having $n \leq 100$ nodes.

**The pros.** The experiments we have performed with the LP heuristic lead us to believe that it is a legitimate competitor for the SIDNEY heuristic, in terms of the Areas of its schedules. Specifically, the means of the Areas of the schedules produced by the LP heuristic were roughly equal to the mean AREAs of the SIDNEY schedules for the same DAGs; see Fig. 2.

This experience suggests that the LP heuristic might be a valuable "auxiliary" scheduler for small DAGs. For instance:

> *We conjecture that the* SIDNEY *heuristic will produce even better schedules if we use the LP heuristic, rather than the* DYNAMIC-GREEDY *heuristic, to schedule its sub-*DAGs*.*

Testing this conjecture is high on our to-do list, as we note in Section 6. At least as importantly is the direct application of the LP formulation of AREA-MAX:

> *When we construct an LP for a* DAG $\mathcal{G}$*, the objective value of the LP serves as a (possibly unachievable) upper bound on* $Area(\mathcal{G})$*, the maximum possible Area of any schedule for* $\mathcal{G}$*.*

**Some perspective.** We illustrate the use of the LP formulation of AREA-MAX as an idealized bound on DAG-Area in Fig. 2 in conjunction with our comparison of the Areas of schedules for small random DAGs produced by the SIDNEY and LP heuristics. As noted earlier, the computational intensiveness of computing the LP has restricted us to small test DAGs, but we do note in the figure that, on the tested DAGs: (a) the SIDNEY and LP heuristics are very close in performance, with a slight advantage to the SIDNEY heuristic; (b) both heuristics produce schedules whose Areas achieve an average of 85% of the LP objective value, hence achieve at least that fraction of $Area(\mathcal{G})$.

## 6 Conclusion

**The past.** The notion of the Area of a schedule for DAGs was introduced in [5], as a proposed mechanism for achieving high performance on the dynamically heterogeneous platforms that are becoming increasingly important. That

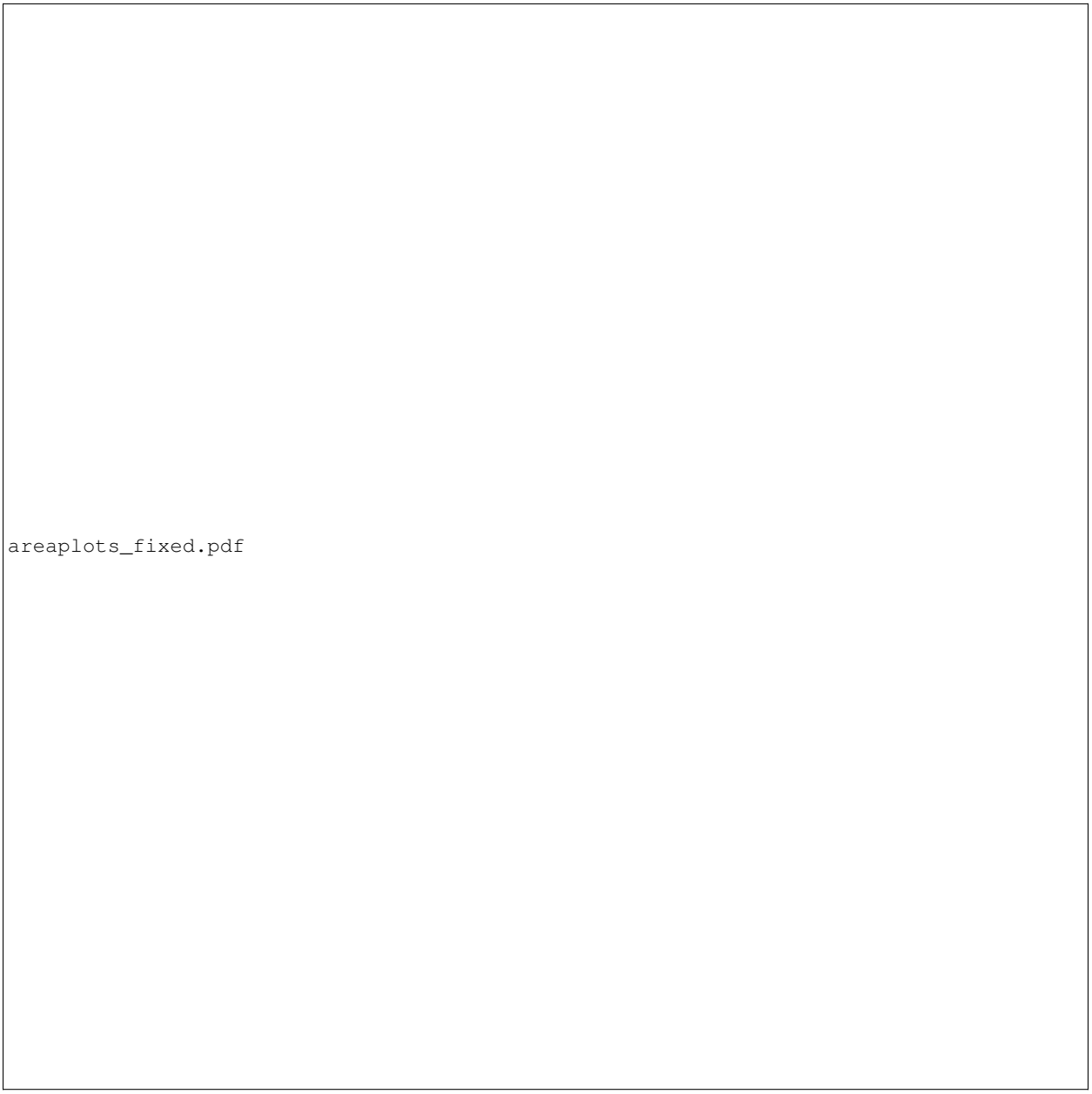


**Fig. 1.** The Area-quality of our heuristic-generated DAG-schedules


**Fig. 2.** Comparing the LP and SIDNEY heuristics via the areas of schedules they produce for random DAGs. The ideal value from the unrounded LP provides perspective.

source developed the basic properties of the quality metric and provided evidence, via simulations, of performance benefits in DAG-schedules that have higher Areas. This evidence, coupled with the apparent complexity of computing Area-optimal DAG-schedules, motivated the development, in [8], of the easily computed AOSPD heuristic, which (1) produced schedules with large Areas and (2) retained much of the performance benefit of Area-optimal schedules [6].

**The present.** The current paper has followed up on two aspects of the earlier studies of Area-oriented DAG-scheduling. (1) We have shown that the observed computational intractability of Area-oriented scheduling is likely inevitable, because achieving Area-optimality is **NP**-complete (Section 3). (2) We have introduced two new polynomial-time Area-oriented scheduling heuristics, the SIDNEY heuristic and the LP heuristic (Section 4). Both produce DAG-schedules whose Areas are, based on extensive simulations, significantly larger than the Areas of schedules produced by the AOSPD heuristic (Section 5).

**The future.** We are actively working on two avenues for extending the work reported here. (1) We are exploring the design of provably good approximation algorithms for finding an AREA-maximum schedule. (2) We are pursuing ways to improve both of our new heuristics, in terms of the efficiency with which they produce DAG-schedules and Areas of the schedules produced. (3) We are initiating studies of the performance benefits of our new heuristics.

*Acknowledgments.* This research was supported in part by US NSF Grant CSR-1217981. The authors are grateful to the following for helpful conversations and valuable advice: Gennaro Cordasco, regarding experimental issues; Greg Malewicz, regarding the LP formulation of AREA-MAX; Michela Taufer and her team, regarding experimental issues.

## References

1. M.A. Bender and C.A. Phillips (2007): Scheduling DAGs on asynchronous processors. *19th ACM Symp. on Parallel Algorithms and Architectures*, 35–45.
2. S.-S. Boutammine, D. Millot, C. Parrot (2006): An adaptive scheduling method for grid computing. *10th Int'l Conf. on Parallel Computing*. In *Lecture Notes in Computer Science 4128*, Springer, Heidelberg, 188–197.
3. H. Casanova, F. Dufossé, Y. Robert, F. Vivien (2011): Scheduling parallel iterative applications on volatile resources. *25th IEEE Int'l Parallel and Distributed Processing Symp.*
4. C. Chekuri, R. Motwani (1999): Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Math. 98(1)*, 29–38.
5. G. Cordasco, R. De Chiara, A.L. Rosenberg (2012): On scheduling DAGs for volatile computing platforms: Area-maximizing schedules. *J. Parallel and Distributed Computing 72*, 1347-1360.
6. G. Cordasco, R. De Chiara, A.L. Rosenberg (2013): An AREA-oriented heuristic for scheduling DAGs on volatile computing platforms. Submitted for publication. See also, Assessing the computational benefits of Area-Oriented DAG-scheduling. *17th Intl Conf. on Parallel Computing*. In *Lecture Notes in Computer Science 6852*, Springer, Heidelberg (2011) pp. I180-I192.
7. 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).*
8. G. Cordasco, A.L. Rosenberg (2013): On scheduling series-parallel DAGs to maximize AREA. Submitted for publication. See also, AREA-optimal schedules for series-parallel DAGs. *16th Int'l Conf. on Parallel Computing (EURO-PAR'10)*. In *Lecture Notes in Computer Science 6272*, Springer, Heidelberg (2010) pp. II380–II392.
9. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein (1999): *Introduction to Algorithms* (2nd Edition). MIT Press, Cambridge, Mass.
10. F. Dong, S.G. Akl (2006): Scheduling algorithms for grid computing: state of the art and open problems. Tech. Rpt. 2006-504, Queen's Univ. School of Computing.
11. T. Estrada, M. Taufer, K.. Reed (2009): Modeling job lifespan delays in volunteer computing projects. *9th IEEE Int'l Symp. on Cluster, Cloud, and Grid Computing (CCGrid).*
12. G. Gallo, M.D. Grigoriadis, and R. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. on Comput.*, 18:30-55,1989.
13. M.R. Garey and D.S. Johnson (1979): *Computers and Intractability*. W.H. Freeman and Co., San Francisco.
14. C. Georgiou, D.R. Kowalski (2011): Performing dynamically injected tasks on processes prone to crashes and restarts. *25th Int'l Conf. on Distributed Computing (DISC'11)*, 165–180.
15. A. González-Escribano, A. van Gemund, V. Cardeñoso-Payo (2002): Mapping unstructured applications into nested parallelism. *High Performance Computing for Computational Science (VECPAR '02)*.
16. R. Hall, A.L. Rosenberg, A. Venkataramani (2007): A comparison of †-scheduling strategies for Internet-based computing. *21st IEEE Int'l Parallel and Distr. Processing Symp.*

17. M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, A. Goldberg (2009): Quincy: fair scheduling for distributed computing clusters. *ACM Symp. on Operating Systems Principles*.
18. D. Kondo, H. Casanova, E. Wing, F. Berman (2002): Models and scheduling mechanisms for global computing applications. *16th Int'l Parallel and Distr. Processing Symp.*
19. E. Korpela, D. Werthimer, D. Anderson, J. Cobb and M. Lebofsky (2000): SETI@home: massively distributed computing for SETI. In *Computing in Science and Engineering* (P.F. Dubois, Ed.) IEEE Computer Soc. Press.
20. Y.-K. Kwok and I. Ahmad (1999): Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys 31*, 406–471.
21. E.L. Lawler (1978): Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Math. 2*, 75–90.
22. M. Lombardi (2013): Robust scheduling of task graphs under execution time uncertainty. *IEEE Trans. Computers 62*, 98–111.
23. D. Millot (2011): Scheduling on unspecified heterogeneous distributed resources. *IEEE Int'l Symp. on Parallel and Distributed Processing: Workshops and Phd Forum (IPDPSW)*, 45–56.
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, A.L. Rosenberg, M. Yurkewych (2006): Toward a theory for scheduling †s in Internet-based computing. *IEEE Trans. Comput. 55*, 757–768.
26. D. Nurmi, R. Wolski, J. Brevik (2005): Model-based checkpoint scheduling for volatile resource environments. *Cluster'2005*.
27. N. Policella (2005): Scheduling with uncertainty: a proactive approach using partial order schedules. *AI Communications 18*, 165–167.
28. A. Radulescu, A.J.C. van Gemund (1999): On the complexity of list scheduling algorithms for distributed memory systems. *13th Int'l Conf. on Supercomputing*, 68–75.
29. A.L. Rosenberg (2004): On scheduling mesh-structured computations for Internet-based computing. *IEEE Trans. Comput. 53*, 1176–1186.
30. V. Sarkar (1989): *Partitioning and Scheduling Parallel Programs for Multiprocessors.* MIT Press, Cambridge, Mass.
31. J.B. Sidney (1975): Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Res. 23(2)*, 283–298.
32. H. Topcuoglu, S. Hariri, M.Y. Wu (2002): Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel and Distributed Systems 13(3)*, 260–274.
33. G.J. Woeginger (2003): On the approximability of average completion time scheduling under precedence constraints. *Discr. Appl. Math. 131(1)*, 237–252.
34. S. Yao and H.-H. S. Lee (2011): Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *IEEE Computer* (Aug, 2011) 55–62.
35. M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica (2008): Improving MapReduce performance in heterogeneous environments. *7th USENIX Symp. on Operating System Design and Implementation*.
36. W. Zheng (2012): A monte-carlo approach for full-ahead stochastic DAG scheduling. *26th IEEE Int'l Parallel and Distributed Processing Symp.: Wkshps. and PhD Forum (IPDPSW)*, 99–112.