A Simpler Primal-Dual Proof of Lawler's Algorithm

Renato Paes Leme * David B. Shmoys †

Abstract

The problem of minimizing the weighted sum of completion times when scheduling jobs on a single machine subject to precedence constraints is NP-hard, but can be solved in polynomial time for interesting families of constraints, such as series-parallel constraints and two-dimensional constraints. A classical decomposition theorem of Sydney leads to a simple polynomial-time algorithm for series-parallel constraints, but this is not particularly efficient since a maximum flow computation is needed to compute the decomposition. In contrast, an efficient $O(n \log n)$ algorithm for scheduling with series-parallel constraints was given by Lawler in 1978, whose proof is very intricate and subtle. Goemans and Williamson gave a primal-dual proof of Lawler's result based on a LP-formulation due to Queyranne and Wang. We give an alternate primal-dual proof which is much simpler and is based on the more natural Chudak-Hochbaum formulation. After that, we turn our attention to two-dimensional precedence constraints, simplifying the recent result by Ambühl and Matrolilli, by proving a stronger structural characterization of a min-cut based approach to compute the optimal solution.

*renatoppl@cs.cornell.edu. Dept. of Computer Science, Cornell University, Ithaca, NY 14853.

†shmoys@cs.cornell.edu. School of ORIE and Dept. of Computer Science, Cornell University, Ithaca, NY 14853.
Research supported partially by NSF grants CCR-0635121, DMI-0500263, DMS-0732196, and CCR-0832782.

1 Introduction

We consider the problem of minimizing the weighted sum of completion times when scheduling jobs on a single machine. This problem is known to be strongly NP-hard; however there are polynomial solutions for special classes of precedence constraints and approximation algorithms for the general case. The latter are based on LP-relaxations of various integer programming formulations of the problem. In this paper, we provide a primal-dual proof of Lawler's algorithm for optimal scheduling with series-parallel precedence constraints that simultaneously (re)proves that there are integral solutions producing feasible schedules for the Chudak-Hochbaum ([CH]) formulation in the series-parallel case. After that, we turn our attention to two-dimensional precedence constraints and simplify a recent polynomial-time algorithm given by Ambühl and Mastrolilli [1]. One reason to focus on this scheduling problem is its close relationship with the vertex cover problem, as highlighted by a number of recent papers [5, 1, 3] on this scheduling problem and its special cases.

Consider a set of n jobs $[n] = \{1, \ldots, n\}$, where each job j has processing time p_j and weight w_j and there is a partial order P on the jobs, which we call precedence constraints. We denote $i \to j$ if i precedes j in the partial order and $i \parallel j$ if the jobs are unrelated. A feasible schedule is a linear extension of the partial order P. The completion time of job j, denoted by C_j, is the sum of the processing times of job j and the jobs scheduled before j. The objective is to minimize $\sum_j w_j C_j$. This was shown to be NP-hard by Lenstra and Rinnooy Kan [12] and Lawler [11]. Lawler also provided a $O(n \log n)$ solution to the case where the precedence constraints are series-parallel. Other polynomial-time algorithms exist for that problem, mainly algorithms based on Sydney's decomposition, but those algorithms involve solving LPs or performing max-flow computations, which makes their running times considerably worse than Lawler's algorithm.

Series-parallel constraints are a class of recursively defined precedence constraints where the empty set of constraints is series-parallel and given two disjoint sets S_1 and S_2 of jobs and series-parallel constraints defined on each of them we can use two operations to produce series-parallel constraints on $S = S_1 \cup S_2$: (i) the series operation: take all the existing precedence relations and $i \to j$ for all $i \in S_1$ and $j \in S_2$ and (ii) the parallel operation: take just the existing precedence relations. Lawler's algorithm has a complicated combinatorial proof which is full of subtleties (see also [10]). A different proof of correctness of Lawler's algorithm was given by Goemans and Williamson [8], who gave a primal-dual proof based on the Queyranne-Wang [15] formulation. This formulation is somewhat unnatural, and this makes the construction intricate and technically hard in some points. Taking $p(S) = \sum_{j \in S} p_j$, the Queyranne-Wang formulation is given by:

$$
\begin{aligned}
&\min \sum_j w_j (M_j + p_j/2) \text{ s.t.} \\
&M_j = C_j - p_j/2, && \forall j \in [n]; \\
&\sum_{j \in S} p_j M_j \geq \frac{1}{2} p(S)^2, && \forall S \subseteq [n]; \\
&\frac{1}{p(B)} \sum_{j \in B} p_j M_j - \frac{1}{p(A)} \sum_{j \in A} p_j M_j \geq \frac{1}{2}(p(A) + p(B)), && \forall A \to B, A \cap B = \emptyset.
\end{aligned}
$$

A more natural formulation of this scheduling problem as an integer linear program was given by Potts [14]. This formulation played a central role in developing the connection between the vertex cover problem and minimizing the weighted completion time with precedence constraints. A further relaxation was proposed by Chudak and Hochbaum [4] and this also helped to highlight the connection between those two problems. We refer to this formulation as [CH]:

$$\min \sum_j w_j \left(p_j + \sum_{i \neq j} \delta_{ij} p_i \right) \text{ s.t.}$$
$$\begin{aligned} &\delta_{ij} + \delta_{ji} = 1, && \forall i, j; \\ &\delta_{ij} = 1, && \forall i \to j; \\ &\delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1, && \forall j \to i, k \parallel i, k \parallel j; \\ &\delta_{ij} \geq 0, && \forall i, j. \end{aligned} \tag{1}$$

The Potts formulation (referred to as [P]) is the same but with the constraint $\delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1$ for all triples (i, j, k). The natural way of thinking about it is that δ_{ij} is a Boolean variable which is 1 if i comes before j and 0 otherwise. Both the relaxation of the natural vertex cover IP and [CH] have half-integral optimal solutions and have the persistence property; that is, there exists an optimal integer solution that can be obtained by "rounding" only those variables set to $\frac{1}{2}$ (i.e., without changing the variables whose LP-value is integral). It is easy to see that a $\{0, 1\}$-feasible point of [P] corresponds to a feasible schedule. The same is not valid for [CH]. However, using the well-known notion of Sidney decomposition ([16]), Correa and Schulz [5] proved that if the precedence constraints are series-parallel, then there is an optimal solution to the [CH] LP that is both integral and corresponds to a feasible schedule.

In this paper, we give a primal-dual proof of the correctness of Lawler's algorithm that also produces a dual solution for the [CH] formulation, thus simultaneously reproving the theorem in [5]. The proof is similar in structure as the proof given by Goemans and Williamson, but as a consequence of the much simpler and natural formulation we used, we get a much simpler proof as well.

A number of papers, starting with Margot, Queyranne and Wang [13] and Chudak and Hochbaum [4], showed that the vertex cover problem and precedence constrained scheduling are closely related problems. Correa and Schulz [5] and Ambühl and Mastrolilli [1] independently showed that precedence constrained scheduling is a special case of the vertex cover problem. This result was in part a consequence of the better understanding of two-dimensional precedence constraints, which are the partial orders that can be written as an intersection of two linear orders. Ambühl and Mastrolilli [1] showed that given an integral solution of [CH] for the two-dimensional partial order (which can be found by a min-cut computation), if this solution doesn't correspond to a feasible schedule, there is an $O(n^3)$ procedure that fixes it, i.e., transforms it to a solution that produces a valid schedule and has the same objective value. We prove that the minimum-cardinality minimum cut always produces a feasible schedule and this yields a much simpler polynomial-time algorithm to compute the optimal schedule for two-dimensional precedence constraints.

Both the vertex cover problem and scheduling with precedence constraints are NP-hard problems for which there are 2-approximations, most of them obtained by studying linear relaxations. Ambühl, Mastrolilli, and Svensson [2] showed that there does not exist a polynomial approximation scheme for the precedence constrained scheduling problem, unless NP is contained in randomized subexponential time. The best hardness of approximation result based on NP-completeness for the vertex cover problem is that no 1.36-approximation algorithm exists (unless P=NP). Khot and Regev ([9]) showed that it is not possible to approximate the optimal vertex cover value to within a factor of $2-\epsilon$ based on the Unique Games Conjecture (which is stronger than assuming that P is not equal to NP). Bansal and Khot [3] recently showed that an even stronger conjecture based on Unique Games would also imply that no approximation guarantee better than 2 is possible for precendence constrained scheduling. Of course, it still remains a distinct possibility that stronger approximation guarantees can be attained for this problem, and that motivates a better structural understanding of its LP formulations.

One further question that remains open is the existence of a primal-dual based approach to the polynomial-time solvability of scheduling subject to two-dimensional precedence constraints. One possible approach to this is to understand a sufficiently rich solvable special case of the vertex cover problem. We believe that our work is a step along the path to achieving this goal.

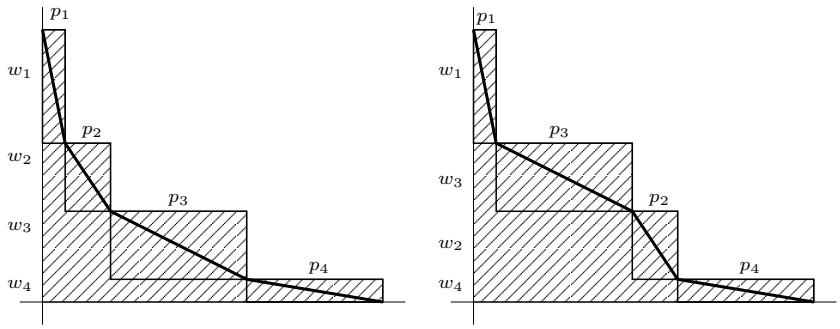

Figure 1: Smith's rule illustrated in a 2D-Gantt chart

2 Primal-dual proof of Lawler's algorithm

Lawler's algorithm One of the oldest results in scheduling theory, due to Smith [17], states that given n independent jobs, a schedule minimizes their total weighted completion time if and only if there is no idle time and the jobs are sorted in order of non-increasing ratio $\rho_j = w_j/p_j$, $j = 1, \ldots, n$. Thus, given precedence constraints, if there is such a sorted order that is consistent with these precedence constraints, then that schedule is also optimal. One way to understand this result is via a 2-dimensional (2-D) Gantt chart, as named by Goemans & Williamson [8], but introduced earlier by Eastman, Even, and Isaacs [7]. In this chart, the x-axis corresponds to time (as in an ordinary Gantt chart), and the y-axis corresponds to the total weight remaining; each job is represented by a rectangle of size p_j by w_j, and, for example if the jobs are scheduled in index order, then the rectangle for job j has its upper left-hand corner at $(\sum_{i=1}^{j-1} p_j, \sum_{i=j}^{n} w_j)$ and its lower-right hand corner at $(\sum_{i=1}^{j} p_j, \sum_{i=j-1}^{n} w_j)$. As shown in Figure 1 below, the "area under the curve" corresponds exactly to the objective function, the slope of the diagonal of the rectangle for job j is $-\rho_j$, and a swap of two consecutive rectangles j and $j+1$ with $\rho_{j-1} < \rho_j$ decreases the objective function.

The essential idea behind Lawler's algorithm for series-parallel constraints is that any such input can be reduced to the problem of scheduling a collection of composite jobs, each of which consists of a disjoint sequence of jobs, such that an optimal (and feasible) schedule can be found simply sorting the composite jobs in the corresponding order; if a composite job α is formed from the simple jobs $j_1, \ldots, j_k$ we say $w_\alpha = w_{j_1} + \cdots + w_{j_k}$, $p_\alpha = p_{j_1} + \ldots + p_{j_k}$, and $\rho_\alpha = w(\alpha)/p(\alpha)$. Consider an instance with series-parallel constraints. These constraints can be represented by a binary *structure tree* where each leaf corresponds to a distinct job and each internal node is either a parallel composition operation or a series composition operation. Given the structure tree, the constraints are obtained the following way: at each internal node, we inherit all of the constraints derived for its two children, and for each series node in which S_1 is the set of leaves in its left subtree, and S_2 is the set of leaves in its right subtree, we also add the constraint $j \to k$ for each job $j \in S_1$ and each job $k \in S_2$.

Lawler's algorithm works *bottom-up* in the structure tree: for each node, it produces a feasible schedule of the jobs in the subtree of that node, or more precisely, it produces a list of composite jobs whose sorting yields a feasible schedule (for the constraints implied by that subtree). Each leaf has a list of one "composite" job (i.e., the single-element composite job corresponding to that leaf). When we process a node, we look at its two children in the structure tree: each has a list of composite jobs sorted by non-increasing ρ-value, which form a feasible schedule. Let them be S_1 and S_2. For a parallel composition node, the algorithm performs a *merge sort* of the S_1 and S_2, sorting them by non-increasing ρ-value. For a series composition node, we would like to do the natural thing: just concatenate S_1 and S_2. The problem is that now the composite jobs need no longer be sorted according to ρ. To fix this, we form a new composite job by concatenating the

smallest ratio job in S_1 to the highest ratio job in S_2; if this new composite job has ratio no more than the lowest remaining one in S_1 and at least the highest ratio in S_2, we are done processing that node; if not, one of these two inequalities is violated, and so we iteratively keep merging this new composite job with the minimum ratio job in S_1 or maximum ratio job in S_2 (depending on which inequality is violated). In fact, as noted in [8], there is an unique minimal way of doing this aggregation of jobs, by taking the lower convex hull in the $2D$-Gantt chart. A more formal description can be found in [10].

We give a simpler new primal-dual proof of optimality by doing the following: we execute Lawler's algorithm and then we look at its execution backwards. While tracing it backwards, we construct a dual solution for the [P] formulation. In the end, we argue that the dual variables corresponding to the equations present in [P] but not in [CH] remained 0, and hence, we have a dual solution to [CH] as well.

Dual of Potts' formulation Potts' formulation [P] is nearly the same as that of Chudak & Hochbaum [CH], but with $\delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1$ (transitivity constraints) for each triple. Consider its dual linear program, where y_{ij} correspond to $\delta_{ij} + \delta_{ji} = 1$, v_{ij} for $\delta_{ij} = 1$ for $i \to j$ and z_{ijk} for the transitivity constraints. In this linear program, we identify the variables y_{ij} and y_{ji} since they refer to the same primal constraint. We also identify z_{ijk}, z_{jki} and z_{kij} since they also refer to the same constraint. The dual to [P] is:

$$\max \sum_{ij} y_{ij} + \sum_{i \to j} v_{ij} + \sum_{ijk} z_{ijk} + \sum_{j} p_j w_j \quad \text{s.t.}$$
$$\begin{cases} y_{ij} + \sum_{k} z_{ijk} \leq p_i w_j & \forall i \nrightarrow j \\ y_{ij} + v_{ij} + \sum_{k} z_{ijk} \leq p_i w_j & \forall i \to j \\ z_{ijk} \geq 0 & \forall i, j, k \\ y_{ij}, w_{ij} \gtrless 0 \end{cases} \tag{2}$$

Consider the output of Lawler's algorithm and suppose that the jobs are indexed in the order in which they are scheduled in this output: the primal solution sets $\delta_{ij} = 1$ if and only if $i < j$. We are aiming to prove that the schedule is optimal via a feasible dual solution, and so we can examine properties of optimal primal-dual pairs of solutions. Of course, a priori there is no assurance that such a solution is achievable - but in fact, this is what we end up proving. By complementary slackness, the dual constraint associated with the variable δ_{ij} should be tight for $i < j$. So the dual solution has the following structure for $i < j$, where we define $\hat{v}_{ij} = v_{ij}$ if $i \to j$ and $\hat{v}_{ij} = 0$ otherwise:

$$\begin{cases} y_{ij} + \hat{v}_{ij} + \sum_{k} z_{ijk} = p_i w_j & \text{(forward constraint)} \\ y_{ij} + \sum_{k} z_{jik} \leq p_j w_i & \text{(backward constraint)} \end{cases} \tag{3}$$

Now, let's see what complementary slackness has to say about the z_{ijk} variables. We view the triple (i, j, k) as a directed cycle. Then there are two possibilities: either we have two arcs forward and one arc backwards, or we have two arcs backwards and one arc forward. If there are two arcs forward, we have $\delta_{ij} + \delta_{jk} + \delta_{ki} = 2$; since the primal constraint is thereby slack, the corresponding dual variable must be 0. In the other case, the primal constraint is tight, so we are allowed to set a positive value for the dual variable.

Suppose we focus on a pair $i < j$. Then we must have $i < k < j$ in order for $z_{ijk} > 0$; if $k < i$ or $k > j$, then we must have $z_{ijk} = 0$. This has a very nice implication: if $i < k < j$ and $z_{ijk} > 0$, this variable appears in the dual constraints for δ_{ij} ($i < j$), δ_{jk} ($j > k$) and δ_{ki} ($k > i$). So, each positive z_{ijk} appears only once in the left-hand-side of a forward constraint and twice in the left-hand-side of a backwards

constraint. This fact is very useful, because the sum of all left-hand-sides of the forward constraints is exactly the objective function, and since they must be tight, the objective function is the sum of the right-hand-side of the forward constraints, plus the constant term $\sum_j p_j w_j$. It follows that if we can satisfy all forward and backward constraints and complementary slackness, then we have a dual solution that matches the primal solution.

Our problem is now reduced to computing values of (y, w, z) that satisfy the equations (3) for $i < j$ so that $z_{ijk} = 0$ for all cycles (i, j, k) with two forward arcs. If we can do that, we are done. Further, notice that this is a really easy thing to do if $i \to j$. For any choice of z_{ijk} values, we can always set:

$$y_{ij} = p_j w_i - \sum_k z_{jik} \quad \text{and} \quad v_{ij} = p_i w_j - y_{ij} - \sum_k z_{ijk},$$

because y_{ij} and v_{ij} are unrestricted in sign. So we can reformulate our problem as satisfying equations (3) for $i < j$ and $i \nrightarrow j$. Notice that this is possible if and only if:

$$\phi_{ij} = p_j w_i - p_i w_j + \sum_k z_{ijk} - \sum_k z_{jik} \geq 0 \quad \forall i < j, i \nrightarrow j.$$

The rest of the proof will be to define values of z_{ijk} so that all ϕ-values become nonnegative. Suppose we perturb the current values of z. What is the influence of increasing z_{ijk}, say by ϵ on the ϕ values? We have that ϕ_{ij} is increased by ϵ and both ϕ_{kj} and ϕ_{ik} are decreased by ϵ.

Notice that we need to guarantee $\phi_{ij} \geq 0$ only when $i \nrightarrow j$. The values of ϕ_{ij} for $i \to j$ can be arbitrary. So, if $k \to j$, we don't care about ϕ_{kj} and changing z_{ijk} means effectively subtracting some value from ϕ_{ik} and adding it to ϕ_{ij}. We summarize this in the following:

Lemma 1 *Consider $i < k < j$. If $i \to k$, it is possible to "transfer" any amount from ϕ_{kj} to ϕ_{ij}; that is, we can decrease the former by $\epsilon > 0$ while increasing the latter by the same amount. Analogously, if $k \to j$ we can "transfer" any amount from ϕ_{ik} to ϕ_{ij}.*

The execution of Lawler's algorithm creates a nested job structure: compound jobs are formed from smaller compound jobs, which are ultimately formed from singletons. When the algorithm terminates, the solution is given in terms of compound jobs that are sorted in non-increasing ρ-order. Our main idea is to solve the problem from the top-level structure to the bottom-level structure.

Given two compound jobs α and β where α is scheduled before β we define: $\phi_{\alpha\beta} = \sum_{i \in \alpha; j \in \beta} \phi_{ij}$. We will design a two-phase procedure to set the z_{ijk} variables, and therefore give a constructive proof to the following theorem:

Theorem 2 *It is always possible to set the the values of z so that in the end, for each pair of simple jobs $i < j$ with $i \nrightarrow j$, we have:*

$$\phi_{ij} = p_i w_j \left(\frac{w_\alpha}{p_\alpha} / \frac{w_\beta}{p_\beta} - 1 \right)$$

where α is the maximal compound job containing i and not containing j and β is the maximal compound job containing j and not containing i.

Corollary 3 *It is always possible to set the the values of z so that in the end, for each pair of simple jobs $i < j$ with $i \nrightarrow j$, we have $\phi_{ij} \geq 0$.*

Proof of Corollary 3 : If α and β are not part of any compound job (i.e. they are in the top level description), then they are in Smith's rule order. Hence, $\frac{w_\alpha}{p_\alpha} \geq \frac{w_\beta}{p_\beta}$ and therefore $\phi_{ij} \geq 0$. Otherwise, they are building

blocks for a compound job. In this case, if they are not in Smith's order, then $\alpha \to \beta$ what is impossible, because, due to the fact that the constraints are series-parallel, it would imply that $i \to j$. ■

Now we begin describing the two-phase procedure. In the first phase, we will only make horizontal transfers: that is, we will transfer from ϕ_{kj} to ϕ_{ij}. This is the same as increasing the value of z_{ijk} for $i < k < j$ and $i \to k$. In the second phase we will make all vertical transfers, that is: we will transfer value from ϕ_{ik} to ϕ_{ij}, what is equivalent to increasing z_{ijk} for $i < k < j$ and $k \to j$. From now on we will just say "transfer" from some ϕ to some other ϕ, but keep in mind that is equivalent to increasing the value of the proper z_{ijk} as indicated here.

First phase: Horizontal Transfers The terms horizontal and vertical transfers come from a geometric interpretation of ϕ_{ij} as rectangular block that is the intersection of the region below job i and to the left of job j. We think of each of those blocks as initialized with $p_j w_i - p_i w_j$ and then we transfer value between them so that in the end the blocks ϕ_{ij} with $i \nrightarrow j$ have a positive value. The rules we defined result in horizontal or vertical transfers.

Let's consider the top-level description of the solution of Lawler's algorithm, when we just have compound jobs sorted according to Smith's rule. In each iteration, we choose one of those compound jobs and break it into its components. This way we get a more "refined" schedule description. In each iteration, we want to set the values of z_{ijk} so to maintain the following invariant:

- for each pair of simple jobs $i < j$ in different compound jobs in the current description such that $i \nrightarrow j$, their ϕ value is given by: $\phi_{ij} = p_j \hat{w}_i - p_i w_j$, where

  $$\hat{w}_i = w_i + p_i \left(\frac{w_\alpha}{p_\alpha} - \frac{w_a}{p_a} \right)$$

  and a is the compound job in the current description containing i, and α is the highest level compound job containing i but not j.

- $z_{ijk} = 0$ for all i, j, k in the same compound job in the current description.

- $z_{ijk} = 0$ for all $i < k < j$ with $k \to j$ (i.e., we don't make vertical transfers)

Consider the following method: if we initialize all $z_{ijk} = 0$, it is easy to see that the invariant holds. For the recursive step: suppose the invariant holds and we pick some compound job and break it into components. Now, for a pair $i < j$, consider the following possible cases:

- they were in the same compound job in the last iteration and now they are in different ones. Then the higher level where they are separated is exactly the compound job that contained them in the current description, so if we just don't change z values that affects ϕ_{ij}, we maintain the invariant.

- a hasn't changed from last iteration, so the invariant remains valid if we don't change z values that affect ϕ_{ij}.

- if a from the previous iteration is split into a sequence of (compound) jobs $a_1, a_2, \ldots, a_k, a_{k+1}, \ldots, a_{k+l}$ where $a_1, a_2, \ldots, a_k$ is in Smith's order and $a_{k+1}, \ldots, a_{k+l}$ is in Smith's order, but the whole sequence is not. We also know that:

  $$\frac{w_{a_{k+1}}}{p_{a_{k+1}}} \geq \cdots \geq \frac{w_{a_{k+l}}}{p_{a_{k+l}}} \geq \frac{w_a}{p_a} \geq \frac{w_{a_1}}{p_{a_1}} \geq \cdots \geq \frac{w_{a_k}}{p_{a_k}}$$

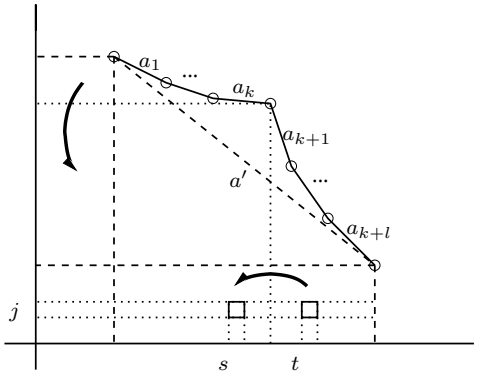

Figure 2: Arrows indicate where transfers occur

and also notice that for $p = 1, \ldots, k$ and $q = k+1, ldots, l$ we have $a_p \to a_q$, so by Lemma 1, we can "transfer" from ϕ_{tj} to ϕ_{sj} for any $s \in a_p$ and $t \in a_q$. We will transfer the following amount (as depicted in Figure 2:

$$\Delta_{st} = p_j p_t \left(\frac{w_{a_q}}{p_{a_q}} - \frac{w_a}{p_a} \right) \frac{p_s \left(\frac{w_a}{p_a} - \frac{w_{a_p}}{p_{a_p}} \right)}{\sum_{p=1\ldots k} \frac{w_a}{p_a} p_{a_p} - w_{a_p}}.$$

Now, let's argue that after those "transfers" the invariant is maintained. For $s \in a_p$, we have that it received:

$$\sum_{t \in a_q; q=k+1,\ldots,k+l} \Delta_{st} = p_j \left(\sum_{q=k+1,\ldots,k+l} w_{a_q} - \frac{w_a}{p_a} p_{a_q} \right) \frac{p_s \left(\frac{w_a}{p_a} - \frac{w_{a_p}}{p_{a_p}} \right)}{\sum_{p=1,ldots,k} \frac{w_a}{p_a} p_{a_p} - w_{a_p}} = p_j p_s \left(\frac{w_a}{p_a} - \frac{w_{a_p}}{p_{a_p}} \right)$$

because:

$$\sum_{q=k+1,ldots,k+l} w_{a_q} - \frac{w_a}{p_a} p_{a_q} = \sum_{p=1,\ldots,k} \frac{w_a}{p_a} p_{a_p} - w_{a_p}.$$

This means that in the value of ϕ_{sj} for some $s \in a_p$ we are updating $\hat{w}_s$ to:

$$w_s + p_s \left(\frac{w_\alpha}{p_\alpha} - \frac{w_a}{p_a} \right) + p_s \left(\frac{w_a}{p_a} - \frac{w_{a_p}}{p_{a_p}} \right) = w_s + p_s \left(\frac{w_\alpha}{p_\alpha} - \frac{w_{a_p}}{p_{a_p}} \right)$$

The calculation to prove that $t \in a_q$ also satisfies the invariant is analogous.

After that, we proved the following lemma (just by setting $a = i$ in the invariant):

Lemma 4 *After some horizontal transfers we reach a situation where for each $i < j$ and $i \nrightarrow j$ we have:*

$$\phi_{ij} = p_j \tilde{w}_i - p_i w_j \quad \textit{where} \quad \tilde{w}_i = p_i \frac{w_\alpha}{p_\alpha} > 0$$

and α is the highest level compound jobs that contains i but not j, where α is well-defined since we view i as a compound job with a single job.

Second phase: Vertical transfers In this phase we do roughly the same thing, but making the transfers in the vertical direction. We will maintain the following invariant:

- for each pair of simple jobs $i < j$ in different compound jobs in the current description such that $i \nrightarrow j$, their ϕ value is given by: $\phi_{ij} = \hat{p}_j \tilde{w}_i - p_i w_j$ where:

$$\hat{p}_j = p_j + w_j \left(\frac{p_\beta}{w_\beta} - \frac{p_b}{w_b} \right)$$

  and b is the compound job in the current description that contains j, and β is the highest level compound job that contains j but not i.

- $z_{ijk} = 0$ for all i, j, k in the same compound job in the current description.

Again, we will begin with the top-level description of the schedule, but with the ϕ values we obtained from the first phase. It is easy to see that the invariant holds for the top-level schedule, because $b = \beta$. For the recursive step: suppose the invariant holds and we pick some compound job and break it into components. Now, for a pair $i < j$, consider the following possible cases:

- they were in the same compound job in the last iteration and now they are in different ones. Then the higher level where they are separated is exactly the compound job that contained them in the current description, so if we just don't change z values that affects ϕ_{ij}, we maintain the invariant.

- b hasn't changed from last iteration, so the invariant remains valid if we don't change z values that affect ϕ_{ij}, the invariant is maintained.

- if b from the previous iteration gets broken in a sequence of (compound) jobs $b_1, b_2, \ldots, b_k, b_{k+1}, \ldots, b_{k+l}$ where $b_1, b_2, \ldots, b_k$ is in Smith's order and $b_{k+1}, \ldots, b_{k+l}$ is in Smith's order, but the whole sequence is not. We also know that:

$$\frac{w_{b_{k+1}}}{p_{b_{k+1}}} \geq \cdots \geq \frac{w_{b_{k+l}}}{p_{b_{k+l}}} \geq \frac{w_b}{p_b} \geq \frac{w_{b_1}}{p_{b_1}} \geq \cdots \geq \frac{w_{b_k}}{p_{b_k}}$$

  and also notice that for $p = 1, \ldots, k$ and $q = k+1, \ldots, l$ we have $b_p \to b_q$, so by Lemma 1, we can "transfer" from ϕ_{is} to ϕ_{it} for any $s \in a_p$ and $t \in a_q$. We will transfer the following amount:

$$\Delta_{st} = \tilde{w}_i w_s \left(\frac{p_{b_p}}{p_{b_p}} - \frac{p_b}{w_b} \right) \frac{w_t \left(\frac{p_b}{w_b} - \frac{p_{b_q}}{w_{b_q}} \right)}{\sum_{q=k+1,\ldots,k+l} \frac{p_b}{w_b} w_{b_q} - p_{b_q}}.$$

  The analysis is identical to that done for the first phase.

And with this, we have given a constructive proof for Theorem 2, which is our main result.

Chudak-Hochbaum formulation and min-cuts Although the result proved in the previous subsection appears to be for the Potts' formulation, in fact it is significantly stronger. The dual solution found is a feasible dual solution for the [CH] formulation, since we just set $z_{ijk} > 0$ for $i < k < j$ and either $i \to k$ or $k \to j$. There is always one precedence relation involved, therefore, we just use dual variables that correspond to constraints that are in the [CH] formulation. This proves that [CH] for series-parallel precedence constraints has a optimal solution which is integral and also a feasible schedule.

For series-parallel constraints (actually, also for the broader class of two-dimensional partial orders) we can write [CH] as a min-cut computation. The LP (1) can be rewritten as:

$$\min \sum_{i \geq j} w_j p_j + \sum_{i<j} \delta_{ij}(p_i w_j - p_j w_i) \qquad \text{s.t.}$$
$$\begin{cases} \delta_{ij} = 1, & \forall i \to j; \\ \delta_{ik} \geq \delta_{jk}, & \forall i \to j < k; \\ \delta_{ki} \leq \delta_{kj}, & \forall k < i \to j; \\ \delta_{ij} \geq 0, & \forall i, j, \end{cases} \tag{4}$$

which can be easily formulated as a min-cut problem. Consider the following graph consisting of one vertex for each δ_{ij} variable with $i < j$ and two extra nodes s and t which will be the source and the sink respectively: (1) for $i \nrightarrow j$, an edge of capacity $p_j w_i$ from s to δ_{ij} and an edge of capacity $p_j w_i$ from δ_{ij} to t; (2) for $i \to j$, an edge of capacity ∞ from s to δ_{ij} and an edge of capacity $p_i w_j$ from δ_{ij} to t; (3) for $i \to j < k$, an edge of capacity ∞ from δ_{jk} to δ_{ik}; (4) for $k < i \to j$, an edge of capacity ∞ from δ_{ki} to δ_{kj}.

It is easy to see that integral feasible values of δ_{ij} correspond to a cut in the graph: associate δ_{ij} with the cut $(S, \overline{S})$ where $S = \{s\} \cup \{\delta_{ij} | \delta_{ij} = 1\}$. It is straightforward to see it is a $1-1$ mapping and that the capacity of the cut is the same of the objective function plus the constant $\sum_j w_j p_j$. So, we just need to prove that the cut obtained by Lawler's algorithm is optimal. In order to do that, we will show how our previous proof produces automatically a flow in the graph matching the capacity of the cut. We say that an linear extension of given precedence constraints is nonseparating if for any pair $i \to j$ and job k that is unrelated to both i and j, then k either precedes i or follows j in the extension.

Theorem 5 *Given the jobs in any non-separating linear extension to the precedence constraints, consider the min-cut graph associated to this instance by the [CH] formulation. Given $\pi : [n] \to [n]$ the ordering of the jobs in the optimal solution produced by Lawler's algorithm, i.e., $\pi(j)$ is the j^{th} job in the ordering, consider the cut $(S, \overline{S})$ where $S = \{s\} \cup \{\delta_{ij}; \pi(i) < \pi(j)\}$. It is the min cut of the graph and the "transfers" made in the first proof produce a max-flow in that graph.*

Proof. As expected, we prove the optimality of that cut by producing a matching flow. We do this the following way:

- for each edge (s, δ_{ij}) and (δ_{ij}, t) add flow corresponding to the capacity of that edge.
- for each transfer $\phi_{\pi(j)\pi(k)}$ to $\phi_{\pi(i)\pi(k)}$ we have $i \to j$, $i \nrightarrow k$ and $\pi(j) < \pi(k)$ (remember i, j, k refer to some non-separating order and $\pi(i), \pi(j), \pi(k)$ relate to the optimal order). Since the jobs are in a non-separating order, there are two possibilities:
  - $j < k$ and therefore both δ_{ik} and δ_{jk} are in the S side, so add the "amount transferred" $\phi_{\pi(j)\pi(k)}$ to $\phi_{\pi(i)\pi(k)}$ to the edge $(\delta_{jk}, \delta_{ik})$;
  - $k < i$ and therefore both δ_{ki} and δ_{kj} are in the T side, so add the "amount transferred" $\phi_{\pi(j)\pi(k)}$ to $\phi_{\pi(i)\pi(k)}$ to the edge $(\delta_{ki}, \delta_{kj})$;
- for each transfer $\phi_{\pi(k)\pi(i)}$ to $\phi_{\pi(k)\pi(j)}$ we have $i \to j$, $k \nrightarrow j$ and $\pi(k) < \pi(i)$. There are two possibilities:
  - $k < i$ and therefore both δ_{ki} and δ_{kj} are in the S side, so add the "amount transferred" $\phi_{\pi(k)\pi(i)}$ to $\phi_{\pi(k)\pi(j)}$ to the edge $(\delta_{ki}, \delta_{kj})$;
  - $j < k$ and therefore both δ_{ik} and δ_{jk} are in the T side, so add the "amount transferred" $\phi_{\pi(k)\pi(i)}$ to $\phi_{\pi(k)\pi(j)}$ to the edge $(\delta_{jk}, \delta_{ik})$;

We don't have exactly a flow, but we have something that can be easily be converted in a max-flow matching the min-cut, since we have: the sum of the flow in the edges crossing the cut from S to $\overline{S}$ matches the min-cut; there is no flow from $\overline{S}$ to S; and the net balance from the nodes δ_{ij} in S is non-negative. It is in fact

$$\text{Net balance}(\delta_{ij}) = p_j w_i - p_i w_j + \text{flows} = p_i w_j \left(\frac{w_\alpha}{p_\alpha} / \frac{w_\beta}{p_\beta} - 1 \right) \geq 0$$

where α is the highest level compound job containing i but not j and β is the highest level compound job containing j but not i. On the other hand, the net balance from the nodes δ_{ij} in $\overline{S}$ is non-positive. It is in fact:

$$\text{Net balance}(\delta_{ij}) = p_j w_i - p_i w_j + \text{flows} = -p_j w_i \left(\frac{w_\beta}{p_\beta} / \frac{w_\alpha}{p_\alpha} - 1 \right) \leq 0.$$

Given that, it is easy to correct the flow so that it becomes a valid flow - we just need to return some flow to s, which can be easily done. Using a procedure similar to the discharge operation in Push-Relabel algorithms, it is easy to return the excess flow to the sink. Notice also that using this, we never change the flow in the edges crossing the cut. ■

3 Two-dimensional precedence constraints: finding the right cut

A two-dimensional partial order is a partial order that can be written as an intersection of two linear orders. An equivalent characterization is given in [6]: they prove that two-dimensional partial orders are the partial orders that have a non-separating linear extension. Let P be a partial order and L be a linear extension of P. We say L is nonseparating if $(i, j) \in P$ and $k \parallel \{i, j\}$ in P implies either $(k, i) \in L$ or $(j, k) \in L$. In particular, this property makes it possible to write [CH] for two-dimensional partial orders as a min-cut problem (as we did with series-parallel constraints). A careful analysis of this kind of constraint can be found in [5]. The authors conjecture that for this type of precedence constraints, [CH] always has an optimal integral solution corresponding to a feasible schedule. This claim is proved by Ambühl and Mastrolilli [1], who give, for any integral solution of [CH] a procedure to transform it in a feasible solution to [P] with the same objective value. The min-cut produced by [CH] might not satisfy the transitivity constraints for triples (i, j, k) such that $i \parallel k \parallel j \parallel i$. When this happens, according to [1], we chose the wrong min-cut. Instead of providing a way to fix the solution, we show how to directly find a min-cut that produces a feasible schedule. This is done by re-interpreting the proof in [1].

Theorem 6 *The minimum-cut with minimum cardinality "source set" always produces a feasible schedule.*

In order to prove Theorem 6, we need to review the results in [1], as well as some facts about the structure of cuts in a graph. First, let's introduce some concepts and notation from [1]: Let $\langle i, j, k \rangle = \{(i, j, k), (j, k, i), (k, i, j)\}$ be a directed 3-cycle of $[n]$ and let $\mathcal{C}$ be the set of all directed 3-cycles. Given a feasible solution δ to [CH] we define $\alpha_{\langle i,j,k \rangle}$ to be 1 if $\delta_{ij} = \delta_{jk} = \delta_{ki} = 1$, and 0 otherwise. This way, $\alpha = \sum_{\langle i,j,k \rangle \in \mathcal{C}} \alpha_{\langle i,j,k \rangle}$ is the number of triples that violate [P]. Given an integral feasible (not necessarily optimal) solution δ to [CH] with $\alpha > 0$ and some job k contained in a violating triple define δ^k to be:

$$\delta^k_{ij} = \begin{cases} 1 - \delta_{ij} & \text{if } \alpha_{\langle ijk \rangle} > 0 \\ \delta_{ij} & \text{otherwise} \end{cases}$$

The following theorem is proved in [1]:

Theorem 7 (Ambühl, Mastrolilli) *Let δ be an integral feasible solution for [CH] that is not feasible for [P], then:*

1. *δ^k is feasible for all k;*

2. *there exists a job k such that the objective value of δ^k is no larger than that of k;*

3. *for any k, $\alpha^k \leq \alpha - |\{(i,j); \alpha_{\langle ijk \rangle} > 0\}|$.*

Our result in based on a stronger version of second item of last theorem (which is Lemma 5(b) in [1]). The authors prove Theorem 7 for any feasible δ. If δ is an optimal solution, then for any k, δ^k is feasible and optimal. This can be easily proved following the proof [1]. The basic idea is: they prove that if $\sum_{(i,j);\alpha_{\langle ijk \rangle}>0} p_j w_i - p_i w_j > 0$ then $\sum_{(ijk)} \alpha_{\langle ijk \rangle} p_k p_i w_j < \sum_{(ijk)} \alpha_{\langle ijk \rangle} p_k p_j \alpha_i$ which contradicts a previous lemma saying it always holds with equality. But since δ is optimal, $\sum_{(i,j):\alpha_{\langle ijk \rangle}>0} p_j w_i - p_i w_j \geq 0$ for all k, and any k for which this inequality holded strictly would imply the inequality from which the result follows. A more formal statement is:

Theorem 8 *Let δ be an integral optimal solution for [CH] that is not feasible for [P]; then for each job k, the objective value of δ^k is equal to that of δ.*

Theorem 8 already gives us an "almost-all-instances" result (that is, a result that holds for a randomly selected instance, except for a set of measure zero): fix the precedence constraints, consider the parameters p, w and suppose that there is more than one minimum cut in the graph; then, one of a polynomial number of equations of the form, the value of δ equals the value of δ_k, must hold. Since there are only finitely many of those equations, the space of p, w for which there is more than one minimum cut is the complement of the union of finitely many polynomials' zero-sets, and therefore, a set of measure zero. To use this result to solve the general case (that is, without the probabilistic assumption), we could add a random perturbation to the p, w parameters and thereby be confident that the solution is unique, while showing that if the perturbation is sufficiently small, it won't affect the optimality of the solution found. There are substantial hurdles to implementing this approach: in particular, even rather sophisticated arguments suggest that the bounds on the perturbation will require a polynomial number of bits per input parameter, which would lead to substantially slower algorithms. Fortunately, there is an easier way of doing that without adding random noise.

Theorem 6 states that the cut with the fewest nodes in the source side always produces a feasible schedule. This cut is actually very simple to compute: given a max-flow in the graph, this cut is the set of reachable nodes from the source in the residual graph. The main ingredient of this proof is that the set of all minimum (s,t)-cuts in a graph form a laminar family, which means that if (S_1, T_1) and (S_2, T_2) are minimum (s,t)-cuts then $(S_1 \cap S_2, T_1 \cup T_2)$ and $(S_1 \cup S_2, T_1 \cap T_2)$ are also. This means that if S is the set of reachable nodes from the source in the residual graph in a max-flow, then $S \subseteq S'$ for all min-cuts (S', T').

Proof of Theorem 6 : Suppose the minimum-cardinality miniumum-cut corresponds to a solution δ with a violated Potts triple (i,j,k). Suppose $i < j < k$; then $\delta_{ij} = \delta_{jk} = 1$ and $\delta_{ik} = 0$, i.e., δ_{ij} and δ_{jk} are on the source side of the cut, and δ_{ik} is on the sink side. Using Theorem 8, there is one min-cut represented by δ^k where $\delta_{ij} = 0$, i.e., δ_{ij} is in the sink-side of the min-cut, which is a contradiction: since it is in the smallest possible min-cut, it must be on the source side of all min-cuts. If $i < k < j$, then the proof is analogous. ■

References

[1] C. Ambühl and M. Mastrolilli. Single machine precedence constrained scheduling is a vertex cover problem. In *ESA*, pages 28–39, 2006.

[2] C. Ambuhl, M. Mastrolilli, and O. Svensson. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In *FOCS '07: Proceedings of the 48th Annual*

IEEE Symposium on Foundations of Computer Science, pages 329–337, Washington, DC, USA, 2007. IEEE Computer Society.

[3] N. Bansal and S. Khot. Optimal long code test with one free bit. In *FOCS*, 2009.

[4] F. Chudak and D. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Oper. Res. Lett.*, 25:199–204, 1999.

[5] J. R. Correa and A. S. Schulz. Single-machine scheduling with precedence constraints. *Math. Oper. Res.*, 30(4):1005–1021, 2005.

[6] B. Dushnik and E. W. Miller. Partially ordered sets. *Amer. J. Math.*, 63:600–610, 1941.

[7] W. Eastman, S. Even, and I. Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management Science*, 11:268–279, 1964.

[8] M. X. Goemans and D. Williamson. Two-dimensional gantt charts and a scheduling algorithm of lawler. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1999.

[9] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-ϵ. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[10] E. Lawler, M. Queyranne, A. Schulz, and D. Shmoys. Weighted sum of completion times. In J. Lenstra and D. Shmoys, editors, *Scheduling*. To appear.

[11] E. L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann. Discrete Math.*, 2:75–90, 1978.

[12] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26:22–35, 1978.

[13] F. Margot, M. Queyranne, and Y. Wang. Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Oper. Res.*, 51(6):981–992, 2003.

[14] C. N. Potts. An algorithm for the single machine sequencing problem with precedence constraints. *Math. Programming Stud.*, 13:78–87, 1980.

[15] M. Queyranne and Y. Wang. Single-machine scheduling polyhedra with precedence constraints. *Math. Oper. Res.*, 16(1):1–20, 1991.

[16] J. B. Sidney. Decomposition algorithms for single machine scheduling with precedence relations and deferral costs. *Oper. Res.*, 23:283–298, 1978.

[17] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.