Precedence Constrained Scheduling in $(2 - \frac{7}{3p+1}) \cdot$ Optimal

Devdatta Gangal *Abhiram Ranade*
Department of Computer Science and Engg.,
Indian Institute of Technology Bombay,
Powai, Mumbai 400076, India.

March 31, 2008

Abstract

We present a polynomial time approximation algorithm for unit time precedence constrained scheduling. Our algorithm guarantees schedules which are at most $\left(2 - \frac{7}{3p+1}\right)$ factor as long as the optimal, where $p > 3$ is the number of processors. This improves upon a long standing bound of $\left(2 - \frac{2}{p}\right)$ due to Coffman and Graham.

1 Introduction

Precedence constrained scheduling is a classical NP-complete problem. It arises in several applications such as project management and parallel computing.

In this paper we consider the simplest variant. In this we are given a directed acyclic graph G, a number p of processors, and a deadline Δ which is an integer. The vertices in the graph represent unit time tasks, and the arcs indicate precedence: if there is an arc from u to v, then task u must be executed before task v. The problem is to decide if the given set of tasks can be executed in Δ time steps using p processors, with each processor being able to execute only one task during any time step. More formally, a length Δ *schedule* for the graph is an assignment of an integer (time slot) $T(v)$ to each vertex v such that (i) $1 \leq T(v) \leq \Delta$, (ii) if u is a predecessor of v in the graph then $T(u) < T(v)$, and (iii) For all i we have $|\{v : T(v) = i\}| \leq p$.

Deciding whether length d schedules exist is known to be NP-complete [8] for variable p. For $p = 2$, Fujii, Kasami and Ninomiya[4] gave a polynomial time algorithm that constructs optimal schedules. Whether the problem is NP-complete for fixed $p \geq 3$ is not known. For the case in which the graph G is a rooted tree, Hu[6] gave a polynomial time algorithm for computing an optimal schedule. Approximation algorithms (those that find schedules which are no longer than a factor α of the optimal schedule length) have also been studied. $2 - \frac{1}{p}$ approximation is easy – any algorithm that will not idle processors while work is available will achieve this. A remarkable algorithm, due to Coffman and Graham[3] was shown to give $2 - \frac{2}{p}$ approximation by Lam and Sethi[7][1]. It is also known[8] that better than $\frac{4}{3}$ factor approximation is not possible unless P=NP.

Remarkably enough, the Coffman-Graham[3] algorithm subsumes the results in [4, 6]. In fact, Coffman and Graham originally proposed their algorithm to solve the 2 processor case; and since it is in fact a refinement of Hu's algorithm, it also optimally solves the case when G is a rooted tree. In some sense, therefore, it may be said that the Coffman-Graham algorithm has been the last word on the problem for over 30 years.

1.1 Main Results

Our first result is an improved approximation algorithm, for $p > 3$. For $p = 2$ our algorithm gives optimal schedules like Coffman-Graham's algorithm, while for $p = 3$ it matches the $2 - \frac{2}{p}$ factor.

Theorem 1 *There exists a polynomial time algorithm which finds a schedule of length at most* $2 - \frac{7}{3p+1}$ *times the optimal, where* $p > 3$ *is the number of processors.*

Our second result is a family of input instances for which the Coffman-Graham algorithm must give schedules of length essentially $2 - 2/p$ for large p.

Theorem 2 *There exists a family of graphs* G_{pq} *such that for all* p*, we have:*

$$\lim_{q \to \infty} \frac{T_{CG}(G_{pq})}{T_{\mathrm{OPT}}(G_{pq})} \;\rightarrow\; 2 - \frac{2}{p}$$

[1] Also see [1] which corrects an error in [7].

where $T_{CG}(G)$ and $T_{\mathrm{OPT}}(G)$ denote the lengths of the schedules produced for a graph G by the Coffman-Graham algorithm and the optimal algorithm respectively.

Proof: G_{pq} consists of vertices labelled u_i, v_i, w_{jk}, for $i = 1, \ldots, q^2$, $j = 0, \ldots, q-1$, $k = 1, \ldots, q(p-2)$. For $i = 1, \ldots, q^2 - 1$ there are edges (u_i, u_{i+1}) and (v_i, v_{i+1}). For $j = 0, \ldots, q-1$ and $k = 1, \ldots, q(p-2)$ there are edges (u_{jq+1}, w_{jk}) and (w_{jk}, v_{jq+3}). We require that $q(p-2)+2$ is a multiple of p, and clearly there are infinitely many choices for this. $T_{CG}(G_{pq}) = (2q - 1 - 2q/p + 2/p)q$, since CG algorithm schedules vertices in decreasing order of the length of the longest path originating at each. But $T_{OPT}(G_{pq}) \leq q(1+q) - 1$. The result then follows. ■

Theorem 2 in fact applies to any algorithm that uses the length of the longest path starting at a vertex v as the scheduling rank of v – discussed below. The best lower bound known previously [2] was $2 - 2/\sqrt{p}$. This bound [2] is in fact applicable to a wider class of algorithms, including the one we will present.

1.2 Outline

In Section 2 we give an overview of the algorithm and the analysis. In Section 3 we present the algorithm. In Section 4 we give the analysis.

2 Overview

Most scheduling algorithms, e.g. [3, 6] begin by computing a *rank* for each node, which indicates its importance – or criticality. The algorithm then simply picks vertices in increasing order of ranks[2] and schedules them at the earliest possible time consistent with the previously scheduled vertices. The ranks often provide lower bounds on the schedule length. Our algorithm also has a similar structure, though one of our key steps involves rearranging previously scheduled vertices.

An important question is how to decide the ranks. For this, we build upon the work of Garey and Johnson[5] on scheduling with deadlines. In this problem, there is the additional requirement that each node v must be scheduled by time $\Delta(v)$ given as a part of the input. A key idea in this work is a procedure for refining the deadlines of each node by propagating deadlines from its ancestors. Garey and Johnson schedule in order of the refined deadlines, and can determine whether deadline respecting schedules exist for $p = 2$.

Although in our problem there are no externally specified deadlines, we begin by specifying a (fictitious) common deadline Δ for all terminal vertices. We then derive deadlines for other vertices, where the notion of a deadline may be formalized as follows.

Definition 1 *An integer d is said to be a deadline for vertex v w.r.t. common deadline Δ iff there is no schedule in which v is scheduled after d and all vertices are scheduled at time $\leq \Delta$.*

[2]The algorithms[3, 6] both schedule in *decreasing* order of the length of the longest path originating at each vertex, differing only in how ties in ranks are broken. Tie breaking is not needed in the example of Theorem 2.

Determining the smallest possible deadline for each vertex is equivalent to finding an optimal schedule, of course. So the key question is how to get close. Our strategy for this is based on the following Lemma. In this, as the rest of the paper, all deadlines are w.r.t. common deadline Δ.

Lemma 1 (Deadline Lemma) *Let v be a vertex and $N \neq \phi$ a set of vertices such that:*

1. *There is a directed path from v to every $u \in N$ having at least $L+1$ edges.*
2. *D is a deadline for every vertex $u \in N$.*

Then $D - L - \left\lceil \frac{|N|}{p} \right\rceil$ is a deadline for v.

Proof: Suppose v is scheduled at step $D - L - \lceil |N|/p \rceil + 1$. Since there is a path of length $L+1$ from v to every $u \in N$, it follows that every $u \in N$ must be scheduled at step $D - \lceil |N|/p \rceil + 2$ or later. Together such vertices u will take at least $\lceil |N|/p \rceil$ slots to fit in. Thus some $u \in N$ must be scheduled at $D+1$ or later. But since D is a deadline for u, it follows that some vertex must be scheduled after Δ. Thus $D - L - \lceil |N|/p \rceil$ is a deadline for v. ■

This Lemma may be used to compute deadlines $D(v)$ for every non-terminal vertex v in reverse topological sort order as follows.

$$D(v) = \min_{L,D} \left\{ D - L - \left\lceil \frac{|N(v,D,L)|}{p} \right\rceil \right\} \tag{1}$$

where $N(v,D,L)$ is the largest set of vertices u, having a path of length at least $L+1$ from v and having $D(u) \leq D$. It is easily seen that this can be done in polynomial time. The maximum range of integer values that $L, D, |N(v,D,L)|$ can take is $|G|$. Hence for every v, the whole process certainly terminates in $|G|^4$. The deadlines used by Garey-Johnson[5] correspond to fixing $L = 0$ in (1) and are thus weaker than ours.

The relationship between deadlines and lowerbounds is straightforward.

Lemma 2 *Suppose that some vertex v in G has deadline δ given global deadline Δ. Then every schedule for G must have length at least $\Delta - \delta + 1$.*

Proof: Suppose a schedule with length $\Delta - \delta$ exists for G, with v scheduled at time $t > 0$. Shifting the schedule ahead by δ, we will have all vertices finishing by Δ and v scheduled at $t + \delta$ which contradicts the fact that v has deadline δ. ■

Our analysis must determine the value of the least deadline assigned to any vertex, for this will establish the strongest lower bound on the schedule length. Estimating this requires an analysis of the schedule constructed by the algorithm. Our arguments for this are much more sophisticated than those in [5], and form the bulk of the rest of this paper.

Main algorithm:

1. Include a dummy vertex $\mathcal{A}$ with edges to all the other vertices. For every terminal vertex v, set $D(v) = \Delta$, where Δ is any number, say 0. Compute deadlines $D(v)$ for all vertices as per equation (1).
2. Order the vertices in increasing order of the deadline, breaking ties arbitrarily.
3. Schedule $\mathcal{A}$ at time 1 on processor 1.
4. for vertex $v = 1 \ldots n$
   (a) RearrangePredecessors(v)
   (b) Let $t =$ earliest time at which v can be scheduled.
   (c) Schedule v in slot t on the smallest numbered free processor.

RearrangePredecessors(v){

1. Let t' be largest such that slot t' contains some predecessor of v.
2. Let $A =$ vertices in slot $t' - 1$ and slot t'.
3. If all vertices in A have the same deadline, and A has at most p predecessors of v, and slot t' has fewer than p vertices
   then
   Move the predecessors of v to slot $t' - 1$, moving out other vertices from slot $t' - 1$ to slot t' as necessary.

}

Figure 1: Scheduling Algorithm

3 Algorithm

For convenience we augment our graph G with a vertex $\mathcal{A}$, with edges to all the other vertices. We fix the global deadline Δ to any fixed number; it should be clear that this value does not affect the relative deadlines.

We will think of the schedule as a 2 dimensional array, with column t indicating the vertices scheduled at time t. We will also refer to column t of the schedule as *slot* t. We will use (t, p) to denote the vertex scheduled in slot t on processor p. For functions f on vertices, we will write $f(t, i)$ instead of $f((t, i))$ for simplicity.

The first step of our algorithm, Figure 1, is to estimate deadlines $D(v)$ for all vertices. We start by setting $D(v) = \Delta$ for all terminal vertices. The deadlines for other vertices are then calculated using equation (1).

We next schedule vertices in non-decreasing deadline order, starting with $\mathcal{A}$ at time 1. It is easily seen that $D(v)$ are *consistent* with precedence, i.e. for any edge (u, v) we have $D(u) < D(v)$.

Thus we are assured that $\mathcal{A}$ has the least deadline and that the predecessors of every vertex v chosen in step 4 (main algorithm, Figure 1) are already scheduled. In step 4(a) we rearrange the predecessors of v if possible. Since vertices in slots $t'-1, t'$ have the same deadline, shuffling them in step 3(a) will not upset precedence constraints, and is equivalent to breaking ties differently in the main algorithm. If all predecessors of v get moved to slot $t'-1$ then v will get scheduled in slot t'. The precise benefit of this rearrangement will be quantified in Lemma 8.

4 Analysis

Let T denote the length of the schedule. A collection of contiguous slots $u, u+1, \ldots, v$ will be called a *region*, and will be written as $[u, v]$. Let $d([u, v])$ be the *deadline drop* for $[u, v]$ defined as $D(v+1, 1) - D(u, 1)$. Slot u will be said to have a deadline drop if $d([u, u]) > 0$. Define $L([u, v]) = v - u + 1$.

The main result follows from three lower bounds we prove on $d([1, T-1])$. The first bound which is implicitly present in [5] is an extension of the familiar "load bound" – the schedule length must be at least the total number of vertices divided by the number of processors. The second bound derives from our deadline estimation procedure (equation 1) and is an extension of the familiar "latency bound" – time must be at least the length of the longest path. The third bound, Lemma 10, is not related to any familiar bounds. We begin by noting some elementary facts about the sequence $\Delta = D(T, 1), D(T-1, 1), \ldots, D(1, 1) = D(\mathcal{A})$.

Lemma 3 *For all t we have $D(t, 1) \leq D(t+1, 1)$. If slot t contains fewer than p vertices, then $D(t, 1) < D(t+1, 1)$, and also every vertex in slot $t' > t$ must be a descendant of some vertex in slot t. If $D(t, 1) = D(t+1, 1)$ then slot t contains p vertices, all of deadline $D(t, 1)$.*

Proof: When some vertex v gets placed in slot $(t+1, 1)$, slot t cannot have been completely empty. At that time, $(t, 1)$ must already have been occupied. Since we schedule in non-decreasing order of deadline $D(t, 1) \leq D(t+1, 1)$. If slot t contains fewer than p vertices, then the only reason a vertex w in slot $t' > t$ cannot be placed in slot t is that some vertex v in slot t is an ancestor of w. Finally, suppose that $D(t, 1) = D(t+1, 1)$. Since the deadlines are the same, the former cannot be an ancestor of the latter. But then, the only reason the latter didnt get placed in slot t must be that slot t is full already, and clearly with nodes of the same deadline. ■

Say a vertex v is *critical* for vertex $(y, 1)$ or for region $[x, y-1]$ if $D(v) \leq D(y, 1)$.

Lemma 4 (Load Bound [5]) *Let n be the number of vertices in $[x+1, y-1]$ critical for $(y, 1)$ and n_x the number of vertices in slot x. Then $d([x, y-1]) \geq \left\lceil \frac{n+1}{pn_x} \right\rceil$ provided $n_x < p$, and $y \leq T$.*

Proof: Since slot x is not full, all vertices in $[x+1, y-1]$ and the vertex $(y, 1)$ must be descendants of some vertex in slot x. Of these, the n that are critical for $(y, 1)$ and $(y, 1)$ itself must have deadline at most $D(y, 1)$. Thus at least $(n+1)/n_x$ of these must be descendants of some vertex u from the n_x vertices in slot x. Thus using equation (1) with $L = 0$, we have $D(u) \leq D(y, 1) - \lceil n + 1/pn_x \rceil$.

But $d([x, y-1]) = D(y,1) - D(x,1) \geq D(y,1) - D(u)$. ■

When applied to region $[1, T-1]$ and using $n_1 = 1$, this lemma essentially yields the familiar bound $d([1, T-1]) \geq$ (Total number of vertices)$/p$.

The above Lemma suggests that we divide the schedule into regions $[x, y-1]$ such that n_x is small; then for each region we are guaranteed a large drop. For $n_x = 1$ we call such regions *blocks*.

For each block we will show a $\left(2 - \frac{7}{3p+1}\right)$ factor bound (Theorem 3), and the main theorem will be proved by summing over the blocks. The block bounds will in turn be obtained by partitioning them into *segments*. A notion similar to blocks (but not segments) is used also in [3, 5].

4.1 Blocks

Let y be the index of the last slot in the schedule. Starting at y scan backwards and determine the largest $x < y$ such that it contains just one vertex critical for $(y, 1)$. Such an x must always exist, since the very first slot only has $\mathcal{A}$. Then region $[x, y-1]$ is called a *block*. If x is not the very first slot, we repeat the procedure scanning backwards from x, identifying more blocks.

While analyzing a block $X = [x, y-1]$, we will only consider the critical vertices in it. Considering critical vertices is equivalent to only considering the schedule till the point when $(y, 1)$ is scheduled, and thus there will be at least one critical vertex in each slot, and Lemma 3 will apply.

Say a slot is full if it has p (critical) vertices, otherwise it is partial.

Lemma 5 (Latency Bound) *Suppose block $X = [x, y-1]$ does not contain a full slot. Then $d(X) \geq L(X)$. If X contains at least one full slot, then $d(X) \geq 1+$ the number of partial slots.*

Proof: From Lemma 3 we know that every partial slot in X has a drop. If X only has partial slots, clearly $d(X) \geq L(X)$. Otherwise suppose slot z is the first full slot in X. If slot z also has a deadline drop then we have the extra drop as needed. So suppose $D(z+1, 1) = D(z, 1)$. By Lemma 3 We know that slot z must contain p vertices, all of deadline $D(z, 1)$. Let A consist of these p vertices and vertex $(z+1, 1)$. Since each slot x through $z-1$ is partial, the vertices in A must have an ancestor in each of these slots. Thus each vertex in A has a path from $(x, 1)$ of length $z - x$ and is a descendant of $(x, 1)$. Using equation (1) with $L = z - x - 1$, we get

$$D(x,1) \leq D(z+1,1) - (z-x-1) - \left\lceil \frac{p+1}{p} \right\rceil = D(z+1,1) - z + x - 1 = D(z+1,1) - L([x,z])$$

Thus $d([x, z]) \geq L([x, z]) = 1+$ the number of partial slots in $[x, z]$. Noting that $d([z+1, y]) \geq$ the number of partial slots in $[z+1, y]$, the result follows. ■

By Lemma 3, we know that the number of partial slots must be at most the length of the longest path in the network. The above Lemma thus shows that the deadline drop (and hence the time) must be at least 1 + the length of the longest path (except for the easy case of when there is no full slot). In this sense it is an extension of the well-known "latency" lower bound. That our bound is stronger by one will be important when blocks are small. Note also that this bound cannot be proved using the deadline definition as in [5] – we need our stronger definition (equation 1).

Before giving further bounds, we first note that already we can match [1, 7]. Let X_i denote the number of slots in X having exactly i (critical) vertices.

Lemma 6 *For any block X, we have $\frac{L(X)}{d(X)} \leq 2 - \frac{2}{p}$.*

Proof: From Lemma 4 we know $d(X) \geq \left\lceil \frac{\sum_{i=1}^{p} iX_i}{p} \right\rceil$. If X has no full slots then from Lemma 5 we know that $d(X) \geq L(X)$ – this assures us an optimal schedule. So we will assume that X does have at least 1 full slot. In which case we know from Lemma 5 that $d(X) \geq 1 + \sum_{i=1}^{p-1} X_i$. Thus:

$$\frac{L(X)}{d(X)} \leq \frac{\sum_{i=1}^{p} X_i}{\max\left(\left\lceil \frac{(\sum_{i=1}^{p} iX_i)}{p} \right\rceil, 1 + \sum_{i=1}^{p-1} X_i\right)}$$

Since we are only considering a single block, we have $X_1 = 1$. For fixed numerator, the above ratio is maximized when $X_i = 0$ for $3 \leq i \leq p-1$. Thus we have

$$\frac{L(X)}{d(X)} \leq \frac{1 + X_2 + X_p}{\max\left(\left\lceil \frac{1+2X_2+pX_p}{p} \right\rceil, 2 + X_2\right)}$$

For $p = 2$, we note that $\left\lceil \frac{1+2X_2+pX_p}{p} \right\rceil = 1 + X_2 + X_p$, giving $L(X)/D(X) \leq 1$. For $p \geq 3$, we drop the ceiling, maximize by equating the two terms in the denominator, and the bound follows. ■

The ratio applies to the entire schedule, since it applies to all blocks. Hence with a much simpler algorithm and analysis we match the result in [1, 7]. We note that the RearrangePredecessors step is not needed in this.

4.2 Segments

The analysis in Lemma 6 is tight when $X_i = 0$ for $3 \leq i \leq p-1$. But if a block only contains slots with 2 vertices or p vertices, it is possible to get a sharper bound. In general, the analysis can be sharpened if the slots in the block are only of certain kinds. Of course, a block will in general contain all kinds of slots; however it is possible to partition it into *segments* whose slots satisfy certain patterns. By deriving bounds for segments, it is possible to get an additional bound (Lemma 10) for the drop for a block. This together with the previous bounds gives us the main result.

From now on we only consider blocks having at least one full slot – for blocks with only partial slots, we know that $d(X) = L(X)$.

Assign labels $1, 2, f, g, b$ to slots based on the number of (critical) vertices and their deadlines as follows. In general for any integer i, an i-slot is a slot with i (critical) vertices. The *first slot* of each block having p vertices (if any) is called an f-slot. A slot with a deadline drop is *good*, one without is *bad*. A good slot which is not an f-slot and has at least 3 vertices will be called a g-slot. A bad slot which is not an f-slot will be called a b-slot. It should be clear that every slot is either a 1-slot, a 2-slot, a g-slot, an f-slot, or a b-slot. If X is a region, we will use X_1, X_2, X_g, X_f, X_b to

Type	Pattern
A1	$1\{2+g\}^*f$
A2	$1\{2+g\}^*f2^+b$
A3	$1\{2+g\}^*f\{g+b\}^*b$
B	$\{g+b\}2^+b$
C	$2\{g+b\}^+b$
D	$2+g$

Table 1: Segment types

denote the number of 1 slots, 2 slots, g-slots, f-slots and b-slots respectively in X. Notice that the total number of slots in X is clearly $X_1 + X_2 + X_g + X_f + X_b$.

Table 1 defines the different kinds of segments, each specified as a regular expression denoting the labelling of its slots. To partition a block into segments, we start at the end of the block, and peel off a segment if its pattern is a suffix of the labelling of the block. We repeat until the entire block is partitioned in this manner.

Lemma 7 *Any block S having at least 1 full slot can be partitioned into segments, the first of which is of type $A1$, $A2$, or $A3$ and the rest of type B, C, D.*

Proof: A block having at least 1 full slot must have the label pattern $1\{2+g\}^*f\{2+g+b\}^*$. In this case we will show that it must satisfy the grammar

$$S \rightarrow A1|A2|A3|SB|SC|SD$$

which immediately proves the Lemma. Figure 2 shows how the patterns can be peeled off scanning S from the end. Starting from the state labelled S at the top, we traverse the edges according to the slot labels. For example, if we encounter a 2-slot, then we follow the leftmost branch and enter the state SD. This state signifies we have encountered a pattern D and that we must resume scanning starting again in state S. $A1, A2, A3$ are terminal states in which the respective patterns have been detected. ■

The important segment types are B, and C. For estimating the drop across segments of type B, we first need to demonstrate the effect of the rearrangement step of the algorithm.

Lemma 8 (B Segment Lemma) *Let $[t, t+k+1]$ be a B segment. Then $d([t, t+k+1]) \geq k+1$.*

Proof: Here we only consider the case $k = 1$ which is indicative. $k > 1$ is left to Appendix A.

Since $k = 1$ the pattern for $[t, t+2]$ is $\{g+b\}2b$. Suppose $d([t, t+2]) = 1$. But since $t+1$ already has a deadline drop, slots t and $t+2$ must be b-slots, i.e. $D(t+1, 1) = D(t, 1)$ and $D(t+3, 1) = D(t+2, 1) = D(t+1, 1) + 1$. Consider the set B of vertices in slot $t+2$ and the vertex $(t+3, 1)$. All these have the same deadline $D(t+2, 1) = D(t+1, 1) + 1$.

If $(t+1, 2)$ is not a predecessor of any vertex in B, then $(t+1, 1)$ must be a predecessor of all vertices in B. But then $D(t+1, 1) \leq D(t+2, 1) - \lceil (p+1)/p \rceil = D(t+1, 1) - 1$. This is a

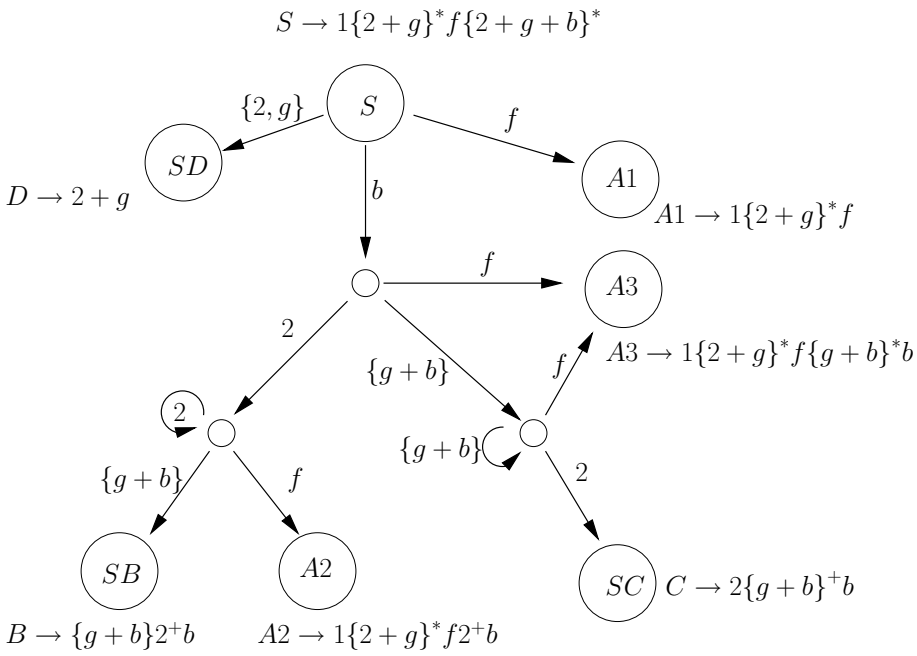

Figure 2: Partitioning into segments

contradiction. Hence $(t+1,2)$ must be a predecessor of some vertex in B. Hence its deadline must also be $D(t+1,1)$.

Suppose some $x \in B$ has at most p predecessors in slots $t, t+1$. Since these predecessors have the same deadline, a rearrangement would have taken place, enabling x to be placed in slot $t+1$. Since this has not happened, every $x \in B$ must have at least $p+1$ predecessors in slots $t, t+1$. So there are a total of $(p+1)^2$ predecessor relations from B to vertices in slots $t, t+1$. These have a total of $p+2$ vertices, and hence at least one, say w must have $(p+1)^2/(p+2) \geq p+1$ successors. Thus $D(w) \leq D(t+2,1) - \lceil p+1/p \rceil = D(t+1,1) - 1$. Giving a contradiction again. ■

Lemma 9 (C Segment Lemma) *Let $X = [u,v]$ be a C segment. Then $d(X) \geq X_2 + X_b/3$.*

Proof: By Lemma 3, $D(w,1) \leq D(v+1,1)$, for $w \leq v+1$. Thus all vertices in b-slots and at least 1 vertex in a g-slot must have deadline at most $D(v+1,1)$. Thus there are at least $pX_b + X_g$ vertices critical for $(v+1,1)$. Hence, by Lemma 4,

$$d(X) \geq \lceil (pX_b + X_g + 1)/2p \rceil \geq \lceil X_b/2 + (X_g+1)/2p \rceil$$

If X_b is even we have $d(X) \geq 1 + X_b/2 \geq X_2 + X_b/3$ as needed. If $X_b = 1$, $X_g \geq 1$ by the definition of the segment, and hence $d(X) \geq X_2 + X_g \geq X_2 + X_b \geq X_2 + X_b/3$. If $X_b \geq 3$ is odd, then we have $d(X) \geq (X_b+1)/2 = X_b/3 + (X_b+3)/6 \geq X_b/3 + 1 = X_2 + X_b/3$. ■

Lemma 10 *For any block X having at least 1 full slot, $d(X) \geq 2 + X_2 + (X_b - 1)/3$.*

Proof: We will abuse X to denote a segment. If a segment X is of any type A, we will prove $d(X) \geq 2 + X_2 + \frac{X_b - 1}{3}$. For segments of type B,C,D we show $d(X) \geq X_2 + \frac{X_b}{3}$. Since every block

consists of one type A segment followed by some B,C,D segments, the result will follow by adding up the segment bounds.

Let the slots of X be $u, \ldots, v$.

B: X is $\{g+b\}2^+b$. By Lemma 8 we have $d(X) \geq X_2 + 1 \geq X_2 + X_b/3$.

C: X is $2\{g+b\}^+b$. Immediate from Lemma 9.

D: X is $\{2+g\}$. Clearly $d(X) = 1 \geq X_2 + X_g \geq X_2 + X_b/3$ since $X_2 \leq 1$ and $X_b = 0$.

A1,A2: Note that Lemma 5 is applicable for prefixes of blocks as well. Thus we have $d(X) \geq 1+$ number of partial slots $\geq 1 + X_1 + X_2 + X_g \geq 2 + X_2 + (X_b - 1)/3$ since $X_1 = 1$ and $X_b \leq 1$.

A3: X is $1\{2+g\}^* f\{g+b\}^+b$. Let the f-slot occur at slot w. As argued in part C above, there are at least $pX_b + X_g + 1$ vertices with deadline at most $D(v+1,1)$, including $(v+1,1)$ itself. Of these at least $pX_b + 1$ have a path of length $w - u \geq X_2$ from $(u,1)$. Thus $D(u,1) \leq D(v+1,1) - X_2 - \left\lceil (p\tilde{X}_b + 1)/p \right\rceil \leq D(v+1,1) - X_2 - X_b - 1$. So,

$$d(X) = D(v+1,1) - D(u,1) \geq 1 + X_2 + X_b \geq 2 + X_2 + (X_b - 1)/3$$ ■

Theorem 3 *For any block X, we have $\frac{L(X)}{d(X)} \leq 2 - \frac{7}{3p+1}$, with the number of processors $p > 3$.*

Proof: If there is no slot in X having p vertices, then we know from Lemma 5 that the schedule is optimal. So we will assume that there is at least 1 slot having p vertices, i.e. $X_f = 1$. The number of partial slots is $X_1 + X_2 + X_g$. Thus by Lemma 5

$$d(X) \quad \geq \quad 1 + X_1 + X_2 + X_g \tag{2}$$

From Lemma 4 we know

$$d(X) \geq \frac{X_1 + 2X_2 + 3X_g + pX_f + pX_b}{p} \tag{3}$$

From Lemma 10 we know

$$d(X) \geq 2 + X_2 + (X_b - 1)/3 \tag{4}$$

Noting that $X_f = 1$, view inequalities (2), (3), (4) as a linear program in variables $X_b \geq 1, X_2 \geq 0, X_g \geq 0$ for fixed $d(X)$, with the objective being

$$\max \quad \frac{L(X)}{d(X)} = \frac{2 + X_2 + X_g + X_b}{d(X)} \tag{5}$$

Solving the linear program in Appendix B we get that the objective is at most $2 - \frac{7}{3p+1}$. ■

Proof of Theorem 1: Let the schedule be made of blocks $X^1, \ldots, X^t$. Then clearly the time for the augmented graphs is $1 + \sum_i L(X^i)$, since the last slot is not accounted in the blocks. The time for the original graph G is thus

$$\sum_i L(X^i) \geq \sum_i \left(2 - \frac{7}{3p+1}\right) d(X^i) = \left(2 - \frac{7}{3p+1}\right)(\Delta - D(\mathcal{A}))$$

But $\Delta - D(\mathcal{A})$ is a lower bound on the schedule length for G. ■

Acknowledgements: We would like to thank Sushant Sachdeva and Shantanu Gangal for proof reading the manuscript.

References

[1] B. Braschi and D. Trystram. A New Insight into the Coffman-Graham Algorithm. *SIAM Journal of Computing*, 23(3):662–669, June 1994.

[2] M. Charikar. Approximation Algorithms for Problems in Combinatorial Optimization, 1995. B. Tech. Project Report, Department of Computer Sc. and Engg., IIT Bombay.

[3] E. Coffman and R. Graham. Optimal Scheduling for Two-Processor Systems. *Acta Informatica*, 1:200–213, 1972.

[4] M. Fujii, T. Kasami, and K. Ninomiya. Optimal sequence of two equivalent processors. *SIAM J. Appl. Math.*, 17(3):784–789, 1969.

[5] M. R. Garey and D. S. Johnson. Scheduling tasks with non-uniform deadlines on two processors. *Journal of the ACM*, 23(3):461–467, 1976.

[6] T. Hu. Parallel Sequencing and Assembly Line Problems. *Operations Research*, 9(6):61–68, November 1961.

[7] S. Lam and R. Sethi. Worst-case analysis of two scheduling algorithms. *SIAM Journal of Computing*, 6:518–536, 1977.

[8] J. Lenstra and A. Rinnooy Kan. Complexity of Scheduling under Precedence Constraints. *Operations Research*, 26:22–35, 1978.

A Proof of Lemma 8

Proof: Assume that $D(t+k+2,1) - D(t,1) \leq k$. But since $t+1,\ldots,t+k$ are 2 slots we know that $D(t+k+1,1) - D(t+1,1) \geq k$. Thus t and $t+k+1$ must be b-slots, and further, the deadlines for vertex 1 in slots $t, t+1, t+2, \ldots, t+i, \ldots, t+k-1, t+k, t+k+1, t+k+2$ must be $D+1, D+1, D+2, \ldots, D+i, \ldots, D+k-1, D+k, D+k+1, D+k+1$ for some D.

Suppose some vertex v in slots $t+1,\ldots,t+k$ is a predecessor of all vertices in slots $t+k+1, t+k+2$. Either v is itself in slot $t+1$, or has a predecessor in slot $t+1$. So w.l.o.g. we may assume that v is in slot $t+1$. Now v has a path of length k to the p vertices in slot $t+k+1$ and vertex $(t+k+2,1)$ all of deadline $D+k+1$. Thus the deadline of v is at most $D+k+1-(k-1)-\lceil (p+1)/p \rceil = D$. But this is a contradiction.

Suppose no single vertex in slots $t+1,\ldots,t+k$ is a predecessor of the vertices in slots $t+k+1, t+k+2$. From this it follows that no vertex in slots $t+i$, $1 \leq i \leq k$ is a predecessor of both vertices in the following slot, and hence each must have at least one successor in the following slot, and hence must have deadline at most $D+i$. Now $(t+1,1)$ and $(t+1,2)$ are also known to have deadline at least $D+1$, and hence their deadline must be exactly $D+1$. Now each of $(t+2,1)$ and $(t+2,2)$ have at least $p+1$ predecessors in slots $t, t+1$ (else rearrangement would have happened), for a total of $2p+2$ relations. Hence at least one vertex from slots $t, t+1$ must be a predecessor of both vertices in slot $t+2$. But now we have a contradiction as before. ■

B Solution of the LP

We first rewrite the LP inequalities (2), (3), (4) by defining $Q = d(X) - 2$ and $Y_b = X_b - 1$.

$$\begin{aligned} X_2 + X_g &\leq Q \\ Y_b + \frac{1 + 2X_2 + 3X_g}{p} &\leq Q \\ X_2 + \frac{Y_b}{3} &\leq Q \end{aligned}$$

The objective (5) may be written as:

$$\max \quad \frac{3 + X_2 + X_g + Y_b}{2 + Q}$$

It is easily seen that all the above inequalities must be tight at the optimum, with $X_2 = \frac{2pQ+3Q+1}{3p+1}$, $X_g = \frac{pQ-2Q-1}{3p+1}$, $Y_b = \frac{3pQ-6Q-3}{3p+1}$. From this the value of the objective function is seen to be

$$2 - \frac{7Q + 3p + 4}{(3p+1)Q + 6p + 2}$$

which is at most $2 - \frac{7}{3p+1}$ for $p \geq 4$, and at most $4/3$ for $p = 3$, noting $Q \geq 1$. Optimality for $p = 2$ follows from Lemma 6.