1. The Man . . .

Ronald Lewis Graham was born on 31 October 1935 in California. His father shifted jobs frequently, moving between the east and west coast of the USA. This required Ron Graham to switch schools frequently. He was often ahead of his peers and allowed to enrol in a higher class in his new school. Graham’s mathematical talents were recognised when he was still quite young. He won a Ford Foundation scholarship and joined the University of Chicago even before he finished high school. He did not take any mathematics courses at Chicago but became an expert at juggling, gymnastics and trampoline, skills he continued to practise at a professional level until late into his life. Once his three-year fellowship ran out, he moved to Berkeley and spent the year 1954–55 enrolled in an electrical engineering course. Here, he took a number theory course given by D. H. Lehmer; though he never took a course with Lehmer again, Graham acknowledged that he learnt the value of independence of thought and an appreciation for the algorithmic issues in mathematics. Graham eventually wrote his PhD thesis under Lehmer’s supervision, but before that, he enlisted in the US Air Force. While posted in Alaska for four years, he continued studying and obtained a BS in physics from the University of Alaska. He returned to Berkeley, obtained an MA in 1961 and his PhD in 1962 for his thesis ‘On Finite Sums of Rational Numbers’. He joined the Bell Telephone Laboratories in New Jersey but continued his research on various aspects of number theory. At a Number Theory Conference in Boulder, Colorado, in 1963, he met Paul Erdős, and from that emerged a

Jaikumar Radhakrishnan is a theoretical computer scientist with research interests in complexity theory, randomness and computation, quantum information and computation, combinatorics, and information theory. After obtaining his PhD in computer science from Rutgers University, USA, in 1991, he joined TIFR, where he is currently a senior professor at the School of Technology and Computer Science, TIFR, Mumbai, and at the International Centre for Theoretical Sciences, Bengaluru.

Keywords
Mathematics, number theory, Graham–Pollak theorem, Graham scan, load balancing, greedy strategy.

*Vol.28, No.9, DOI: https://doi.org/10.1007/s12045-023-1670-9
Figure 1. Two of Ron Graham’s highly influential books.

legendary friendship, which arguably proved to be life-defining for Graham. He developed a formidable reputation as a problem solver and made foundational contributions to areas such as approximation algorithms, computational geometry, and (jointly with his wife Fan Chung) quasi-randomness.

He held academic positions, taught at several universities, and wrote several very influential books, most notably *Ramsey Theory* (with Bruce Rothschild and Joel Spencer, 1980) and *Concrete Mathematics* (with Donald Knuth and Oren Patashnik, 1989). As Director of Information Sciences at Bell Labs (1962–95) and later as Chief Scientist (1996–99), Graham established and led an outstanding group of mathematicians and computer scientists whose pioneering work still informs much of current research in discrete mathematics and theoretical computer science. In this period, he also held several prestigious visiting and part-time academic positions; from 1999, he held the Irwin and Joan Jacobs Endowed Chair of Computer and Information Science at the University of California at San Diego. Graham’s contributions were recognised by numerous awards and memberships of several academies. He was President of the American Mathematical Society in 1993–
94 and President of the Mathematical Association of America in 2003–05. On receiving the Steele Award in 2003, Graham said:

“I can’t remember a time when I didn’t love doing mathematics, and that desire has not dimmed over the years (yet!). But I also get great pleasure sharing mathematical discoveries and insights with others, even though this can present a special challenge for mathematicians talking to non-mathematicians. However, I really believe that this type of communication will become increasingly important in the future.”

When Ron Graham passed away on 06 July 2020, the mathematical world lost one of its most charismatic personalities and one of its foremost ambassadors

2. Glimpses of His Mathematics . . .

In this section, we present three examples from Graham’s work that bear his stamp and name. Clearly, these examples do not do justice to the wide breadth of Graham’s contributions—they do not touch upon number theory and Ramsey theory, two of his favourites. The selection is skewed, leaning more towards algorithmic issues. But as stated above, this was something that Graham talked about himself. The examples below require no specialized knowledge, and most of the presentation should be accessible to a student in high school.

2.1 The Graham–Pollak Theorem (1971–72)

The following problem arose in Bell Labs in connection with an abstract routing problem in telephone networks. The problem was to assign to each node \( v \) of a connected network a code word \( c_v \) of \( \ell \) symbols from the set \{0, 1, \*\}, so that the distance in the network between nodes \( v \) and \( w \) is the number of positions where \( c_v \) and \( c_w \) differ, that is, one has a zero and one has a 1. There is a

---

1 The following website maintained by Fan Chung has links to several interesting documents and videos on various aspects of Ron Graham’s life: https://mathweb.ucsd.edu/ fan/ron/
route-finding scheme based on such an assignment that efficiently establishes telephone connections between distant nodes in the network along the shortest path. For several networks, Graham and Pollak determine the minimum length \( \ell \) for which such an assignment of code words exists; several problems remain open [1]. The importance of the original motivation for this problem has receded, but its resolution for the special case of the complete network on \( n \) nodes remains one of the most charming applications of algebra to a problem in combinatorics. We will present one of the versions of this argument; Suhail Sherif [2], has a beautiful YouTube video explaining this argument with pictures.

For a network \( G \), let \( \ell(G) \) be the minimum length \( \ell \) so that there is an assignment of code words from \( \{0, 1, *\}^\ell \) satisfying the above property. Let \( K_n \) denote the complete network with vertex set \( \{1, 2, \ldots, n\} \).

**Theorem 1** (Graham and Pollak [3, 4]). \( \ell(K_n) = n - 1 \).

**Proof.** Suppose there is an assignment \( (c_v : v = 1, 2, \ldots, n) \) of code words with length \( \ell \) to the vertices of \( K_n \). We wish to show that \( \ell \geq n - 1 \). For \( k = 1, 2, \ldots, \ell \), we define sets of vertices

\[
L_k = \{v : c_v[k] = 0\} \quad \text{and} \quad R_k = \{v : c_v[k] = 1\}.
\]

We will construct \( \ell \) polynomials on variables \( X_1, X_2, \ldots, X_n \) using these sets: for \( k = 1, 2, \ldots, \ell \), let

\[
P_k(X_1, X_2, \ldots, X_n) = \left( \sum_{i \in L_k} X_i \right) \left( \sum_{j \in R_k} X_j \right).
\]

Note that the monomial \( X_i X_j \) \( (i < j) \) appears in \( P_k \) precisely when \( c_i[k] \) and \( c_j[k] \) are both non-star and different; so \( X_i X_j \) appears in precisely one of the polynomials \( P_1, P_2, \ldots, P_\ell \). Thus,

\[
\sum_{k=1}^{\ell} \left( \sum_{i \in L_k} X_i \right) \left( \sum_{j \in R_k} X_j \right) = \sum_{1 \leq i < j \leq n} X_i X_j;
\]

that is,

\[
X_1^2 + X_2^2 + \cdots + X_n^2 = \left( \sum_{i=1}^{n} X_i \right)^2 - 2 \sum_{k=1}^{\ell} \left( \sum_{i \in L_k} X_i \right) \left( \sum_{j \in R_k} X_j \right). \tag{2}
\]
Let us suppose that \( \ell \leq n - 2 \) and derive a contradiction. Consider the following system of \( \ell + 1 \) linear equations in \( n \) variables:

\[
\sum_{j \in L_k} X_k = 0 \quad \text{for } k = 1, 2, \ldots, \ell; \quad (3)
\]

\[
X_1 + X_2 + \ldots + X_n = 0. \quad (4)
\]

Since \( \ell \leq n - 2 \), the number of equations, namely \( \ell + 1 \), is at most \( n - 1 \); hence, this system has a non-zero solution, \( X_1 = \alpha_1, X_2 = \alpha_2, \ldots, \alpha_n \), where the \( \alpha_i \) are rational numbers. If we substitute \( \alpha_i \) for \( X_i \) in (2), we obtain \( 0 < \text{LHS} \) and \( \text{RHS} = 0 \)—a contradiction.

The above proof is remarkable for it unexpectedly appeals to algebraic notions to arrive at the conclusion. Several other proofs are known for the Graham–Pollak theorem, but no purely combinatorial proof is known. We mention the following insightful counting argument due to Sundar Vishwanathan [5], which obtains the required non-zero solution for the system of equations (3)–(4) using a direct counting argument, thereby removing at least one of the uses of algebra in the proof.

Let \( K \) be a large integer and consider assignments

\( \beta = (\beta_1, \beta_2, \ldots, \beta_n) \in \{1, 2, \ldots, K\}^n \)

to the variables \( X_1, X_2, \ldots, X_n \). There are \( K^n \) such assignments. Each LHS in the system evaluates to a value in \( \{1, 2, \ldots, Kn\} \) under this evaluation. If \( K^n/(Kn)^{n-1} > 1 \), then there are distinct assignments \( \beta \) and \( \beta' \) such that the LHS of each equation in the system evaluates to the same value under \( \beta \) and under \( \beta' \). Then, we set \( \alpha_i = \beta_i - \beta'_i \) for \( i = 1, 2, \ldots, n \), and obtain the required non-zero solution to the system (3)–(4).

2.2 The Graham Scan (1972)

A classical algorithm due to Ronald Graham for constructing the convex hull of a set of points of the form \((x, y)\) in the two-dimensional
A classical algorithm due to Ronald Graham for constructing the convex hull of a set of points of the form \((x, y)\) in the two-dimensional plane (i.e., \(\mathbb{R}^2\)), is a fundamental contribution to the area of computational geometry. Recall that the convex hull of a set of points \(P \subseteq \mathbb{R}^2\) (assume they do not all reside on the same line) is the smallest convex polygon that contains all the points of \(P\). We then have the following natural algorithmic question.

Given a set \(P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2\) of \(n\) points, list the vertices of the convex hull in the order in which they appear in the boundary of the hull, say in the anticlockwise order.

Ronald Graham’s famous algorithm, now popularly known under the name Graham Scan, solves this problem in \(O(n \log n)\) time, assuming that real numbers can be multiplied and compared in unit time.

\textit{The Algorithm}

Perhaps the first algorithm that one thinks of for the problem proceeds as follows. Say we start at the point \(a_0 \in P\) with the minimum \(x\)-coordinate (if there is a tie, choose the point among the contenders with the smallest \(y\)-coordinate), and identify an edge \((a_0, a_1)\) of the convex hull, so that \((a_0, a_1)\) makes the smallest angle (in the range \((-\pi, \pi]\)) with the \(x\)-axis. Now, we move to \(a_1\) and find the point \(a_2\) such that \((a_1, a_2)\) makes the smallest angle with the direction \((a_0, a_1)\). We repeat this process, and to identify the next edge on the boundary; we stop when we return to \(a_0\). The chain of successive edges thus identified grows, wrapping around the set of points \(P\). The algorithm is sometimes referred to as the gift wrapping algorithm, and sometimes as Jarvis march, after its author R. A. Jarvis [6]. A straightforward implementation would take time \(O(nh)\), where \(h\) is the number of vertices in the convex hull; in the worst case, it might take \(\Omega(n^2)\) time. The Wikipedia page for the gift wrapping algorithm [7] contains a nice animation of this algorithm\(^2\). Note that for each new edge we find, we examined essentially all the vertices; that is, for each edge we start from scratch, abandoning all the information we learnt while we were looking for the previous edge.

\(^2\)The animation finds the vertices in the clockwise order.
Graham’s algorithm departs from the gift-wrapping strategy in two crucial ways: (i) at an intermediate stage, the part of the boundary currently identified is allowed to include vertices that are not on the boundary—the algorithm deletes these spurious vertices at a later stage; (ii) the algorithm scans the vertices in a convenient order so that the addition and deletion from the chain can be performed efficiently. Here is how this is done. As before, we start from the leftmost point $a_0$, which we know must be a vertex of the convex hull. We then examine the remaining points and see if they must be added to the boundary. Graham’s great insight was to scan the remaining vertices in a specific order. Imagine that the origin is located at $a_0$; consider the angles that the lines that connect $a_0$ to the other points make with the x-axis; let $(p_1, p_2, \ldots, p_{n-1})$ be an ordering of the remaining points so that the angle that $(a_0, p_i)$ makes with the x-axis is non-decreasing. We are now ready to identify the vertices of the convex hull, which we maintain as chain $a = (a_0, a_1, \ldots, a_r)$ ($a$ is actually a list, but we use a different word to avoid confusion); initially the chain is $a = (a_0)$. In each iteration, we examine the next vertex $p_i$, and determine if it must be added to $a$. At the end of the iteration, the chain $a$ will contain the vertices of the convex hull of the points $(a_0, p_1, \ldots, p_i)$; the vertices of the hull will appear in the anticlockwise order. Note, however, that with each point $p_i$ we process, the chain $a$ might grow (by at most one when $p_i$ is added to the chain) or shrink (when several previous vertices in the chain need to be removed to accommodate the new vertex $p_i$). We will soon describe precisely how this is done. It will turn out that the insertions and deletions happen only at the right end of $a$. Furthermore, a vertex is added to the chain only once and hence can be deleted at most once.

When a vertex $p_i$ is examined, it is added to the chain. In a later step, if $p_i$ is found not to be a vertex of the convex hull, it might be removed, so clearly in all at most $n - 1$ vertices are ever removed. Determining if a vertex needs to be removed requires a small calculation, which can be carried out in constant time. It, therefore, takes $O(n)$ time to identify the vertices of the hull af-
ter the sorted list is available. The running time of the Graham scan is thus dominated by the time taken to sort the list and is $O(n \log n)$.

**Input:** A list of points $P$. We assume the points are distinct and not all on a common line.

**Step 1:** (Initialization) Find the anchor point $a_0$. Remove $a_0$ from the list of points. Let the chain be $a = (a_0)$ initially.

**Step 2:** (Ordering the points) Sort the remaining points to obtain $(p_1, p_2, \ldots, p_{n-1})$, so that the angle the line segment $(a_0, p_i)$ makes with the $x$-axis is non-decreasing as $i$ goes from 1 to $n - 1$. If there are ties, list the point that is farther from $a_0$ later. Add $a_0$ as $p_n$ at the end of the list.

**Step 3:** (The scan) For $i = 1, 2, \ldots, n$, examine the point $p_i$ and adjust the current chain $a$ so that it is the list of vertices of the convex hull of $(p_1, p_2, \ldots, p_i)$.

3.1 If the chain has only one element add $p_i$ to the chain.

3.2 Else, let $(a_{r-1}, a_r)$ be the last two vertices of the current chain. If the path $a_{r-1} -- a_r -- p_i$ makes right turn at $a_r$, remove $a_r$ from the chain $a$, and go to Step 3.1, else add $p_i$ to $a$ and go to Step 3.

Some care is needed in coding up the algorithm; Gries and Stojmenović [8] point out that several works that offered improvements or variations of Graham’s original algorithm had errors (usually minor). We will not present a formal justification for the algorithm’s correctness, but we will make the following remarks.

**Finding the anchor point:** The anchor point is intuitively the leftmost point in $P$ (the one with the minimum $x$-coordinate); if there is a tie, we take $a_0$ to be the point with the lowest $y$-coordinate.

**Sorting:** We state above that the points are to be sorted based on angles; again, we assume that the angles lie in the range $(-\pi, \pi]$. Determining the angles between lines typically involves the inverse of trigonometric functions and is subject to errors due to
round-off. Note, however, that we do not need to determine the angles precisely; we only need to compare them. This can be achieved more directly. Suppose we wish to determine which of \((a_0, p_i)\) and \((a_0, p_j)\) makes a bigger angle with the \(x\)-axis. Let \(p_i - a_0 = (x, y)\) and \(p_j - a_0 = (x', y')\). We compute the determinant

\[
D = \begin{vmatrix} x & x' \\ y & y' \end{vmatrix} = xy' - yx'.
\]

We say that \((a_0, p_j)\) makes a greater angle then \((a_0, p_i)\) if \(D > 0\), breaking ties first using the \(x\)-coordinate and then, if the tie persists, using the \(y\)-coordinate. More precisely, to take care of ties, we place \(p_j\) after \(p_i\) in the sorted list if the tuple \((D, x' - x, y' - y) > (0, 0, 0)\) in the lexicographic ordering. So if \(p_i\) and \(p_j\) make the same angle at \(a_0\) with the \(x\)-direction, then we place \(p_j\) later if it is farther from \(a_0\).

**When is it a right turn?** In Step 3.2, the algorithm needs to determine if the path \(a_{r-1} - a_r - p_i\) makes a right turn at \(a_r\). We compare the angles \(a_r - a_{r-1} = (x, y)\) and \(p_i - a_{r-1} = (x', y')\) make with the \(x\)-axis by computing the determinant \(D\) exactly as above. Now, if \(D \leq 0\), then we say that the path makes a right turn at \(a_r\). In particular, if going from \(a_r\) to \(p_i\) amounts to proceeding from \(a_r\) further in the same direction as \(a_{r-1}\) to \(a_r\), we still consider it a right turn.

As do most algorithms in computational geometry, Graham scan looks rather interesting to visualize and animate, e.g., see the Wikipedia page for Graham scan [9].

### 2.3 Load Balancing (1966, 1969)

Ron Graham’s early works [10, 11] on scheduling can arguably be considered the beginning of the modern worst-case analysis of approximation algorithms; the competitive analysis of online algorithms can be seen to have their origins in these works, although the term was coined several decades later [12]. In this section, we review Graham’s celebrated theorems on scheduling\(^3\). Consider the following problem.

\[^3\text{We essentially follow the presentation in Chvátal’s [13] lecture notes.}\]
There are $n$ jobs (tasks) $J_1, J_2, \ldots, J_n$, and $m$ machines $M_1, M_2, \ldots, M_m$. The machines have identical capabilities; in particular, any job can be assigned to any machine. Each job has a processing time; the processing time of job $J_i$ is $p_i$. Given $I = \langle m, n, (p_i : i = 1, \ldots, n) \rangle$ as input, we would like to assign the jobs to the machines so that the entire set of jobs is completed as early as possible. That is, we wish to find an assignment $\sigma : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, m\}$ such that

$$\max_j \sum_{i : \sigma(i) = j} p_i$$

is as small as possible. Note that we care only about the time taken to finish all the jobs, not when a particular job is finished. Let $\text{Opt}(I)$ be the time the best assignment takes to complete all the jobs; that is,

$$\text{Opt}(I) = \min_{\sigma} \max_j \sum_{i : \sigma(i) = j} p_i,$$

where the minimum is taken over all assignments $\sigma$.

**The Greedy Strategy**

To arrive at the assignment, we can follow the following greedy strategy. We scan the list of jobs in the order $J_1, J_2, \ldots, J_n$. After the first $i - 1$ jobs $J_1, \ldots, J_{i-1}$ have been assigned to processors, we determine the loads on each machine $M_j$, that is, the sum of the processing times of all the jobs assigned to that machine so far. We assign job $J_{i+1}$ to a processor that has the least cumulative load at that point (say, we choose the first processor if there is a tie). Let $\sigma_G(I)$ be the assignment that this strategy produces for the input $I$, and let

$$\text{Greedy}(I) = \max_j \sum_{i : \sigma_G(i) = j} p_i.$$

**Theorem 2** (10). For all inputs $I$ with $m$ processors, we have

$$\text{Greedy}(I) \leq \left(2 - \frac{1}{m}\right) \text{Opt}(I).$$
Proof. Suppose the machine \( M_{j^*} \) has the maximum load under the greedy assignment \( \sigma_G \). We wish to show that

\[
\sum_{i: \sigma_G(i) = j^*} p_i \leq \left(2 - \frac{1}{m}\right) \text{Opt}(I).
\]

Suppose \( J_{k^*} \) is the last job that was assigned to \( M_{j^*} \). When \( J_{k^*} \) was assigned to \( M_{j^*} \) it had the least load, which at that point was precisely

\[
\sum_{i: \sigma_G(i) = j^*, i \neq k^*} p_i.
\]

Thus the total load of all the jobs other than \( J_{k^*} \) is at least \( m \) times this quantity; thus,

\[
\sum_i p_i \geq m \sum_{i: \sigma_G(i) = j^*, i \neq k^*} p_i + p_{k^*} \quad (6)
\]

\[
= m \text{Greedy}(I) - (m - 1)p_{k^*}. \quad (7)
\]

Since the maximum load must be at least the average load, we have \( \text{Opt}(I) \geq \frac{1}{m} \sum_i p_i \), that is,

\[
m \text{Opt}(I) \geq \sum_i p_i. \quad (8)
\]

Combining this with (7), we obtain

\[
m \text{Greedy}(I) \leq m \text{Opt}(I) + (m - 1)p_{k^*}, \quad (9)
\]

that is,

\[
\text{Greedy}(I) \leq \text{Opt}(I) + \left(1 - \frac{1}{m}\right) p_{k^*}. \quad (10)
\]

This inequality tells us that the amount by which the greedy algorithm is worse than the optimum can be bounded in terms of the processing time of the job that finishes last: so, all will be well if it ends well. We will make use of this insight again. For now, we observe that (5) follows from this because \( \text{Opt}(I) \geq p_{k^*}. \quad \square \)
The above algorithm worked under the assumption that the processors could work on the jobs independently. In general, however, we might have precedence constraints among the jobs of the form $J_i - J_j$, which says that $J_j$ can be started only after $J_i$ is completed. The input $I$ now has the form $(m, n, (p_i : i = 1, \ldots, n), C)$, where $C$ is the set of precedence constraints. We now need to find the precise schedule for each job: for each job $J_i$, along with the processor it runs on, we must specify the interval of time when that job is processed on that processor. As a natural extension of the above greedy strategy, we may construct the schedule using a queue of jobs that are ready to be performed; initially, we add to the queue the jobs that have no prerequisites. At each step, we remove the job from the queue and schedule it appropriately on a machine in the earliest time slot available, respecting all the constraints; then, we consider jobs that have not been put in the queue so far and insert them into the queue all such jobs all of whose prerequisites have been assigned slots already. Let $\text{Queue}(I)$ be the time it takes for all the jobs processed under the schedule produced by this algorithm for the input $I$; let $\text{Opt}(I)$ be the corresponding time taken for the best assignment of slots. An analysis very similar to the one presented above for the greedy algorithm shows the following.

**Theorem 3** (Graham [10]).

$$\text{Queue}(I) \leq \text{Opt}(I) \left( 2 - \frac{1}{m} \right). \tag{11}$$

**Proof.** We only sketch the argument. Let $F$ be the time taken by the above algorithm to process $J$. Suppose $J_k$ is the last job that is completed. Why was $J_k$ delayed so much? Here is the crucial observation: there is a chain of precedences $C = J_{i_1} - J_{i_2} - \ldots - J_{i_r} = J_k$, such that at every time instant when some processor is idle, one of the jobs in the chain is being processed on some other processor. That is, at any instant when no processor from $C$ is being processed, every other processor is busy with jobs not in $C$. We then have the following inequalities (the first is the same as (8), the second is corresponds to the inequality $\text{Opt}(I) \geq p_k$ that we
used at the last proof, the third corresponds to (7)):

$$\text{OPT}(I) \geq \frac{1}{m} \sum p_i;$$

$$\text{OPT}(I) \geq \sum_{\ell=1}^{r} p_{i\ell};$$

$$\text{QUEUE}(I) \leq \frac{1}{m} \sum p_i + \left(1 - \frac{1}{m}\right) \sum_{\ell=1}^{r} p_{i\ell}.$$

Our claim (11) immediately follows from these. □

It is natural to wonder whether theorem 2 and theorem 3 offer the best possible guarantees for their respective approaches. They do. Consider an input $I_0$ where there are $m - 1$ jobs, each taking $m - 1$ units of time, another $m - 1$ jobs taking 1 unit of time, and finally, one job that takes $m$ units of time. Assume that there are no precedence constraints. If the jobs are processed greedily in the order in which they are presented, then $\text{GREEDY}(I_0) = 2m - 1$ and $\text{OPT}(I_0) = m$. In particular, $\text{GREEDY}(I_0) = (2 - \frac{1}{m})\text{OPT}(I_0)$. This example suggests that perhaps, whenever possible, we should schedule longer jobs before shorter jobs. We now assume that there are no precedence constraints, so we are free to consider the jobs in a convenient order. Consider the following strategy. Reorder the jobs so that their processing times are non-increasing, and then consider the jobs in this order and assign them following the greedy strategy: each job is assigned to a processor whose cumulative load arising from the previous jobs assigned to it is the minimum. Let $\text{ORDGREEDY}(I)$ be the time taken to complete all the jobs if they are assigned using the above strategy.

**Theorem 4** (Graham [11]). *For all inputs $I$ with $m$ processors (and no precedence constraints), we have*

$$\text{ORDGREEDY}(I) \leq \left(\frac{4}{3} - \frac{1}{3m}\right)\text{OPT}(I).$$

We will base our proof on the following key observation$^4$.  

$^4$The author as a graduate student at Rutgers University learnt this from Ravi Boppana.
Lemma 1. If there is an optimum schedule for $I$ in which no machine processes more than two jobs, then $\text{OrdGreedy}(I) = \text{Opt}(I)$.

Proof. Suppose the processing times for the jobs in non-increasing order are:

$$p_1, p_2, \ldots, p_k,$$

where $n = k \leq 2m$. If $k < 2m$, consider a new input $I'$ obtained by adding $2m-k$ dummy jobs $J_{k+1}, J_{k+2}, \ldots, J_{2m}$, with processing times $p_{k+1}, p_{k+2}, \ldots, p_{2m} = 0$. Clearly,

$$\text{OrdGreedy}(I) = \text{OrdGreedy}(I') = \max_{\ell} p_{\ell} + p_{2m-\ell+1}.$$

So our claim will follow if we show that for $\ell = 1, 2, \ldots, m$,

$$\text{Opt}(I') \geq p_{\ell} + p_{2m-\ell+1}.$$

By starting with the optimum schedule for $I$ and assigning the dummy jobs to the machines that have only one job assigned to them, we obtain an optimum schedule for $I'$ where every processor processes exactly two jobs: the $2m$ jobs get paired-up into $m$ disjoint pairs. There are $\ell$ jobs in the list $J_1, J_2, \ldots, J_\ell$, and only $\ell - 1$ jobs in the list $J_{2m}, J_{2m-1}, \ldots, J_{2m-\ell+2}$. So a job in $J_1, J_2, \ldots, J_\ell$ must be paired up with a job in $J_{2m}, J_{2m-1}, \ldots, J_{2m-\ell+1}$. Thus

$$\text{Opt}(I') \geq \min\{p_1, p_2, \ldots, p_{\ell}\} + \min\{p_1, p_2, \ldots, p_{2m-\ell+1}\} = p_\ell + p_{2m-\ell+1}.$$

Thus, the ordered greedy algorithm under our assumption produces an optimal schedule.

Proof of theorem 4. Consider the assignment produced by the ordered greedy algorithm. Suppose the jobs listed in non-decreasing order of their processing times are $J_1, J_2, \ldots, J_n$, and in the assignment produced by the ordered greedy algorithm, the last job to finish is $J_{k^*}$. We have two cases. First suppose $p_{k^*} > \text{Opt}(I)/3$. Restrict attention to the input $I^*$ consisting of the first $k^*$ jobs. Since $\text{Opt}(I^*) \leq \text{Opt}(I)$, no processor processes more than two
jobs in any optimal assignment for $I^*$. From lemma 1, we have that

$$\text{OrdGreedy}(I) = \text{OrdGreedy}(I^*) = \text{Opt}(I^*) \leq \text{Opt}(I).$$

On the other hand, if $p_{k^*} \leq \text{Opt}(I)/3$, we conclude using (10) that

$$\text{OrdGreedy}(I) \leq \text{Opt}(I) + \left(\frac{m-1}{m}\right)\text{Opt}(I)/3 = \left(\frac{4}{3} - \frac{1}{3m}\right)\text{Opt}(I).$$

\[\square\]

There are inputs for which the inequality in theorem 4 is tight. Consider the input $I_1$ with $m$ machines and $n = 2m + 1$ jobs with processing times

$$2m - 1, 2m - 1, \ldots, m + 2, m + 2, m + 1, m + 1, m, m, m.$$

Then, $\text{OrdGreedy}(I_1) = 4m - 1$ and $\text{Opt}(I_1) = 3m$ (how?), so that

$$\text{OrdGreedy}(I_1) = \left(\frac{4}{3} - \frac{1}{3m}\right)\text{Opt}(I_1).$$

Acknowledgment

I am grateful to Mohit Garg for reading this article carefully and for his very helpful suggestions.

Suggested Reading


