Approximability and nonapproximability results for minimizing total flow time on a single machine

Citation for published version (APA):

Document status and date:
Published: 01/01/1995

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Approximability and Nonapproximability Results for Minimizing Total Flow Time on a Single Machine

Dedicated to the memory of Gene Lawler

HANS KELLERER †  THOMAS TAUTENHABN ‡  GERHARD J. WOEINGER §

Abstract

We consider the problem of scheduling $n$ jobs that are released over time on a single machine in order to minimize the total flow time. This problem is well-known to be NP-complete, and the best polynomial time approximation algorithms constructed so far had (more or less trivial) worst-case performance guarantees of $O(n)$. In this paper, we present one positive and one negative result on polynomial time approximations for the minimum total flow time problem: The positive result is the first approximation algorithm with a sublinear worst-case performance guarantee of $O(\sqrt{n})$. This algorithm is based on resolving the preemptions of the corresponding optimum preemptive schedule. The performance guarantee of our approximation algorithm is not far from best possible as our second, negative, result demonstrates: Unless $P = NP$, no polynomial time approximation algorithm for minimum total flow time can have a worst-case performance guarantee of $O(n^{\frac{2}{3} - \varepsilon})$ for any $\varepsilon > 0$.

Keywords: scheduling, approximation algorithm, worst-case analysis, total flow time, release time, single machine.

1 Introduction

Scheduling independent jobs on a single machine has been extensively studied under various objective functions. We consider one of the basic problems of this kind, which from a worst-case point of view also appeared to be among the most intractable ones: There are given $n$ independent jobs $J_1, \ldots, J_n$ which have to be scheduled nonpreemptively on a single machine. Each job $J_i$ has a processing time $p_i$ and becomes available for execution at its release time $r_i$. All job data are known in advance. Without loss of generality we assume that the smallest release time is equal to zero. In a certain schedule for these jobs, let $C_i$ be the completion time of job $J_i$, let $S_i$ be...
its \textit{starting time} (i.e., $S_i + p_i = C_i$) and let $F_i = C_i - r_i$ denote its \textit{flow time}, respectively. The objective is to determine a schedule that minimizes the total flow time $\sum F_i$. This problem is commonly denoted by $1|r_i|\sum F_i$.

In case all job release times are identical, the problem can be solved in $O(n \log n)$ time by applying the well-known \textit{Shortest Processing Time} (SPT) rule, see Smith [18]. For arbitrary release times, the problem becomes NP-complete (Lenstra, Rinnooy Kan and Brucker [15]). Several authors (Chandra [4], Chu [6], Deogun [7], and Dessouky and Deogun [8]) developed branch-and-bound algorithms for $1|r_i|\sum F_i$. Other papers (Bianco and Ricciardelli [3], Dyer and Wolsey [10], Hariri and Potts [13], and Posner [17]) gave branch-and-bound algorithms for $1|r_i|\sum w_i F_i$, the problem of minimizing the total \textit{weighted} flow time. Gazmuri [12] designed asymptotically optimal algorithms for minimizing total flow time under very general probability distributions of the job data.

The preemptive version $1|\text{pmtn}, r_i|\sum F_i$ of the problem can be solved in polynomial time by the \textit{Shortest Remaining Processing Time} (SRPT) rule, see e.g., Baker [2]. The objective value of the optimum preemptive schedule clearly is a lower bound on the objective value of the nonpreemptive problem. Ahmadi and Bagchi [1] have shown that it dominates all other known lower bounds for $1|r_i|\sum F_i$.

\textbf{Approximation algorithms.} For an instance $I$ of $1|r_i|\sum F_i$, let $F^H(I)$ denote the total flow time obtained when algorithm $H$ is applied to $I$ and let $F^*(I)$ be the objective value of an optimum solution for $I$. We usually write $F^H$ and $F^*$ instead of $F^H(I)$ and $F^*(I)$, respectively, if the instance $I$ is clear from the context. Now let $\rho : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ and let $H$ be an approximation algorithm. We say that algorithm $H$ has \textit{worst-case performance guarantee} $\rho$ if

\[ \sup \{ F^H(I)/F^*(I) \mid I \text{ is an instance with } n \text{ jobs} \} \leq \rho(n) \]

holds for all integers $n \geq 1$.

Chu [5] and Mao and Rifkin [16] investigated several ‘reasonable’ algorithms for $1|r_i|\sum F_i$, but none of them yielded a sublinear worst-case performance guarantee. E.g., a straightforward procedure for $1|r_i|\sum F_i$ is the \textit{Earliest Start Time} (EST) algorithm: “Whenever the machine becomes free for assignment, take the shortest of the (at that time) available jobs and assign it to the machine”. The following example shows that the total flow time in an EST-schedule can be a factor of $n$ away from the optimum: Let $\varepsilon$ be a very small positive real number. Then choose $p_1 = 1$, $p_2 = \ldots = p_n = \varepsilon$, $r_1 = 0$, $r_2 = \ldots = r_n = \varepsilon$. The EST-schedule processes the jobs in the ordering $(1, 2, \ldots, n)$ with total flow time $\Omega(n)$, whereas an optimum schedule processes the jobs in the ordering $(2, 3, \ldots, n-1, 1)$ with total flow time $O(1)$.

Another direct approach is the \textit{Earliest Completion Time} (ECT) rule. An ECT-schedule is formed by scheduling jobs without any unnecessary idle time where the next job to be scheduled is the one with the earliest possible completion time of all unscheduled jobs. Again, let $\varepsilon$ be a very small positive real number. Then the job data $p_1 = 1 + 2\varepsilon$, $p_2 = \ldots = p_n = \varepsilon$ and $r_i = i - 1$ for all $i = 1, \ldots, n$ yields an ECT-schedule of $(2, 3, \ldots, n-1, 1)$ with total flow time $\Omega(n)$, whereas the optimum schedule is $(1, 2, \ldots, n)$ with total flow time $O(1)$. Again, we get a worst-case ratio of $n$.

\textbf{New Results.} The results of this paper are as follows. First we present a polynomial time algorithm with worst-case performance guarantee of $O(\sqrt{n})$. The algorithm starts from an optimum solution of the corresponding \textit{preemptive} instance. Then the preemptive schedule is step by step
transformed into a nonpreemptive one by successively resolving the preemptions. Our proof also demonstrates that for any instance, the minimum total flow time of an optimum nonpreemptive schedule is at most a factor of $\sqrt{n}$ larger than the corresponding minimum total flow time of the optimum preemptive schedule.

In the second part of the paper, we prove that polynomial time approximation algorithms for minimum total flow time on a single machine cannot have a worst-case performance guarantee of $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$. This result is derived by an appropriate reduction from the NP-complete Numerical 3-Dimensional Matching problem.

**Organization of the paper.** Section 2 describes the approximation algorithm and proves the claimed worst-case performance guarantee of $O(\sqrt{n})$. Section 3 gives the non-approximability result and Section 4 contains the discussion.

## 2 The Approximation Algorithm

In this section, we will design a polynomial time approximation algorithm for the minimum total flow time problem that has worst-case performance guarantee $O(\sqrt{n})$. The main idea is to start with an optimum solution of the corresponding problem where preemption is allowed, and then to get rid of the preemptions while increasing the total flow time by only some ‘moderate’ amount.

As already mentioned in the introduction, an optimum preemptive schedule can be obtained by applying the shortest remaining processing time (SRPT) rule which is defined as follows: The remaining processing time $P_i(t)$ of job $J_i$ is the amount of processing time of $J_i$ which has not been scheduled before time $t$. At any time $t$, the available job $J_i$ with shortest remaining processing time $P_i(t)$ is processed until it is either completed or until another job $J_j$ with $p_j < P_i(t_j)$ becomes available. In the second case, $J_i$ is preempted and $J_j$ is processed.

We denote the total flow time in an optimum preemptive schedule by $F^*_{pmtn}$. With every job $J_i$, we associate the interval $I_i = [S_i, C_i]$ in the preemptive schedule. Note that due to possible preemptions, the length of $I_i$ in general need not be equal to $p_i$. Let us observe some simple properties of the SRPT-schedule: If a job $J_i$ has a preemption at time $t$, then the processing of some job $J_j$ with $p_j < P_i(t_j)$ starts at this time. Consequently, job $J_i$ cannot return to the machine until job $J_j$ is completed.

**Observation 2.1** In the SRPT-schedule, for any two jobs $J_i$ and $J_j$ the corresponding intervals $I_i$ and $I_j$ are either disjoint or one of them contains the other one. Furthermore, there is no machine idle time during $I_i$ and $I_j$.

With the preemptive schedule we associate in the following way a directed ordered forest that describes the containment relations of the intervals $I_i$. The jobs $J_1, \ldots, J_n$ form the vertices of the forest. We introduce a directed edge going from job $J_i$ to $J_j$ if and only if $I_j \subseteq I_i$ and there does not exist a job $J_k$ with $I_j \subseteq I_k \subseteq I_i$. For every vertex $J_i$, its sons are ordered from left to right according to the ordering of their corresponding intervals $I_j$. This yields a collection of ordered directed out-trees. Out-trees that consist of a single vertex are called trivial out-trees. To complete the definition of the forest, we also order the roots of these out-trees from left to right according to the ordering of their corresponding intervals. By $T(i)$ we denote the maximal subtree of this forest rooted at $J_i$ (hence, $T(i)$ exactly contains those jobs that are processed during $[S_i, C_i]$). A leaf is a job with out-degree zero (hence, it does not have any preemptions).
Since our approximation algorithm transforms the preemptive schedule into a nonpreemptive one, the forest associated with the final schedule will be a collection of $n$ trivial out-trees. In the following three Sections 2.1, 2.2 and 2.3, we describe three procedures for removing preemptions and simplifying the forest structure. Every procedure acts on a certain subtree $T(i)$ and removes all preemptions of the job $J_i$. Section 2.4 explains how the approximation algorithm applies and combines these three procedures.

2.1 How to handle Small Subtrees

A subtree $T(i)$ is called small if it contains at most $\sqrt{n}$ jobs. In this case we use the following procedure to resolve the preemptions of job $J_i$ and of all jobs contained in $T(i)$.

**Procedure** SmallSubtree($i$)

Let $S_i = S_{i_0} < S_{i_1} < \cdots < S_{i_k}$ denote the starting times of the jobs in $T(i)$ in the current preemptive schedule. Remove all jobs and reinsert them without preemptions in the ordering $J_i, J_{i_1}, J_{i_2}, \ldots, J_{i_k}$.

(For an illustration, see Figure 1). It is easy to see that SmallSubtree($i$) does not decrease the starting time of any job. Hence, the resulting schedule still obeys all release times. Completion times of jobs outside of $T(i)$ are not changed, and the completion times of jobs in $T(i)$ are all increased by less than $C_i - r_i$. Since there are at most $\sqrt{n}$ jobs in $T(i)$,

$$\Delta_{\text{small}}(i) = \sqrt{n}F_i$$

is an upper bound on the increase of the objective function caused by SmallSubtree($i$).

2.2 How to handle the Last Root

Now let $J_i$ be the root of the rightmost non-trivial out-tree (i.e., $J_i$ is preempted, but all jobs that are processed after $C_i$ are without preemptions). Such a root $J_i$ is called the last root and may be handled according to the following procedure.

**Procedure** LastRoot($i$)

Compute $\lfloor \sqrt{n} \rfloor + 1$ time points $t_0, \ldots, t_{\lfloor \sqrt{n} \rfloor}$ such that there are exactly $h \cdot p_k$ units of idle time between $C_i$ and $t_h$ in the current schedule for all $h \in \{0, \ldots, \sqrt{n}\}$. Determine $0 \leq k \leq \lfloor \sqrt{n} \rfloor - 1$ so as to minimize the value of $k + \{|j : t_k \leq S_j \leq t_{k+1}\}$.
In case there is a job processed during \([t_k, t_k + p_i]\), shift it to the right until its processing is disjoint from \([t_k, t_k + p_i]\). If necessary shift also some later jobs to the right, but without changing their relative orderings and without introducing any unnecessary idle time. Then remove \(J_i\) and reschedule it in the interval \([t_k, t_k + p_i]\).

(For an illustration, see Figure 2). Since the job starting times are not decreased, the resulting schedule is again feasible. Moreover, by definition of the numbers \(t_h\), there are exactly \(p_i\) units of idle time between \(t_k\) and \(t_{k+1}\). Hence the shifting of jobs takes place only within the interval \([t_k, t_{k+1}]\) and no jobs outside of this interval are affected by \(\text{LastRoot}(i)\).

**Lemma 2.2** The value of \(k\) determined by Procedure \(\text{LastRoot}(i)\) satisfies

\[
k + |\{j : t_k \leq S_j \leq t_{k+1}\}| \leq \frac{3}{2} \sqrt{n}.
\]

**Proof.** Define \(n_h = |\{j : t_k \leq S_j \leq t_{h+1}\}|. \) Then

\[
\sum_{h=0}^{[\sqrt{n}] - 1} (h + n_h) \leq \frac{1}{2} \sqrt{n} (\sqrt{n} - 1) + n \leq \frac{3}{2} \sqrt{n} \sqrt{n}.
\]

Since \(k\) was chosen to minimize \(k + n_k\), the value \(k + n_k\) is bounded from above by the average of these \(\sqrt{n}\) numbers.

Procedure \(\text{LastRoot}(i)\) increases the flow time of job \(J_i\) by at most \(kp_i + \sum_{j=1}^{n} p_j\). Furthermore, the flow times of at most \(n_h = |\{j : t_k \leq S_j \leq t_{h+1}\}|\) other jobs are increased by at most \(p_i\). By applying Lemma 2.2 one gets that \(\text{LastRoot}(i)\) increases the value of the objective function by at most

\[
\Delta_{last}(i) = kp_i + \sum_{j=1}^{n} p_j + n_k p_i \leq \frac{3}{2} \sqrt{n} p_i + \sum_{j=1}^{n} p_j.
\]

(2)

Finally observe that in the new schedule \(J_i\) is processed entirely after its completion time in the old schedule.

![Figure 2: Illustration for Procedure LastRoot(i) with \(i = 1\).](image)


2.3 How to handle Fathers and Sons

The procedure FatherSon(i, j) described below may only be applied if the jobs $J_i$ and $J_j$ fulfill the following four conditions.

(C1) $J_j$ is a son of $J_i$.
(C2) All sons of $J_i$ in $T(i)$ that lie to the right of $J_j$ are leaves (i.e. jobs without preemption).
(C3) There are less than $\sqrt{n}$ jobs that lie to the right of $J_j$ in $T(i)$.
(C4) Just before executing FatherSon(i, j), job $J_i$ is removed and rescheduled entirely after its old completion time.

Condition (C4) may be fulfilled e.g. since $J_i$ is rescheduled by LastRoot(i) or (as we will see) since it is rescheduled by FatherSon(k, i) where $k$ was the father of $i$ at this time. Condition (C4) has the following consequences: At the moment as SRPT decided to start processing $J_j$ instead of $J_i$, it did so because the remaining processing time of $J_i$ was larger than $p_j$. Hence, the total time that $J_i$ is processed during $[C_j, C_i]$ is at least $p_j$ and in case $J_i$ is moved to some place after $C_i$, there is sufficient empty space in $[C_j, C_i]$ to schedule all of $J_j$.

Procedure FatherSon(i, j)

All jobs that are processed during $[C_j, C_j + p_j]$ are shifted to the right until their processing is disjoint from $[C_j, C_j + p_j]$. If necessary, also some later jobs are shifted to the right, but without changing their relative orderings and without introducing any unnecessary idle time.

Then remove $J_j$ and reschedule it to the interval $[C_j, C_j + p_j]$.

(For an illustration, see Figure 3). Note that in the new schedule $J_j$ is processed entirely after its completion time in the old schedule. Furthermore note that FatherSon(i, j) transforms all sons of $J_i$ in $T(i)$ that lie to the right of $J_j$ into trivial out-trees.

Procedure FatherSon(i, j) produces another feasible schedule where job $J_j$ has no preemption any more. Only jobs in the interval $[C_j, C_j]$ are shifted to the right and no other jobs are affected. All the shifted jobs are contained in $T(i)$, lie to the right of $J_j$ and by condition (C3), their number is less than $\sqrt{n}$. The completion time of every moved job (including job $J_j$) increases by at most $p_j$. Hence the total increase in the value of the objective function that FatherSon(i, j) produces can be bounded by

$$\Delta_{\text{fson}}(j) = \sqrt{n} p_j.$$ (3)

2.4 Putting everything together

Finally we describe our approximation algorithm in detail. The algorithm starts with the optimum schedule for the preemptive problem and determines the corresponding directed ordered forest.

In the first phase, procedure SmallSubtree(i) is applied to all jobs $J_i$ for which $2 \leq |T(i)| \leq \sqrt{n}$ holds. Afterwards, every subtree $T(i)$ either fulfills $|T(i)| = 1$ (and $J_i$ has no preemption) or $|T(i)| > \sqrt{n}$ holds.

In the second phase, the algorithm goes through a number of steps and repeatedly modifies the rightmost non-trivial out-tree. Every such step increases the number of trivial out-trees by
Minimizing the Total Flowtime

Figure 3: Illustration for Procedure FatherSon\( (i, j) \) with \( i = 1 \) and \( j = 2 \).

at least \( \sqrt{n} \). Every step consists of one call to LastRoot possibly followed by several calls to FatherSon. For a non-leaf vertex \( J_i \) in the forest, let \( \text{rmnls}(i) \) denote the index of its rightmost non-leaf son and let \( \text{leaf}(i) \) denote the number of sons of \( J_i \) lying to the right of \( \text{rmnls}(i) \) (obviously, all these \( \text{leaf}(i) \) sons must be leaves). Every step of the second phase is performed as follows. Let \( J_i \) be the root of the rightmost non-trivial out-tree.

\[
\begin{align*}
\text{LastRoot}(i); \\
\text{While } \text{rmnls}(i) \text{ exists and } \text{leaf}(i) < \sqrt{n} \text{ do} \\
& \quad \text{FatherSon}(i, \text{rmnls}(i)) \\
& \quad i := \text{rmnls}(i) \\
\text{EndWhile}
\end{align*}
\]

Let \( J_{i^*} \) denote the last job whose preemptions are resolved in such a step. Then either \( J_{i^*} \) has more than \( \sqrt{n} \) sons (that all are leaves) to the right of \( \text{rmnls}(i^*) \), or job \( \text{rmnls}(i^*) \) does not exist. In the latter case all sons of \( J_{i^*} \) are leaves and, according to the result of the first phase, their number must be greater than \( \sqrt{n} \). In either case, there are at least \( \sqrt{n} \) leaves that formerly were preemptsing \( J_{i^*} \), the father of \( J_{i^*} \), its grandfather, and so on, the whole chain of ancestors up to the formerly ‘last root’. Since the execution of the step removes all preemptions from this chain of ancestors, it turns all these at least \( \sqrt{n} \) leaves into trivial out-trees.

In the second phase of the algorithm the above described step is repeated over and over again, until all preemptions have been removed. Since every step produces at least \( \sqrt{n} \) new trivial out-trees, the second phase terminates after at most \( \sqrt{n} \) steps. This completes the description of our approximation algorithm.

**Theorem 2.3** The above described approximation algorithm for the problem \( 1|\tau_i|\sum F_i \) has a worst-case performance guarantee of \( O(\sqrt{n}) \) and it can be implemented to run in \( O(n^{3/2}\log n) \) time. Moreover, there exist instances for which the algorithm yields a schedule whose objective value is \( \Omega(\sqrt{n}) \) away from the optimum.

**Proof.** To analyze the worst-case behaviour of the algorithm, we define job classes \( S, L \) and \( F \). Class \( S \) contains the jobs \( J_i \) for which SmallSubtree\( (i) \) is executed, \( L \) the jobs \( J_i \) for which LastRoot\( (i) \) is executed, and \( F \) the jobs \( J_f \) for which FatherSon\( (i, j) \) is executed. Since every procedure removes all preemptions of the corresponding job, every job is contained in at most one of the classes \( S, L \) and \( F \). Moreover, \( |L| \leq \sqrt{n} \) holds, since LastRoot\( (i) \) is executed exactly once per step in the second phase, and there are at most \( \sqrt{n} \) steps.
Minimizing the Total Flowtime

The constructed nonpreemptive schedule has an objective value $F^H$ which satisfies

$$F^H \leq F^*_\text{pmtn} + \sum_{i \in S} \Delta_{\text{small}}(i) + \sum_{i \in L} \Delta_{\text{last}}(i) + \sum_{i \in F} \Delta_{\text{son}}(i)$$

$$\leq F^*_\text{pmtn} + \sum_{i \in S} \sqrt{n} F_i + \sum_{i \in L} \left(\frac{3}{2} \sqrt{n} p_i + \sum_{j=1}^{n} p_j\right) + \sum_{i \in F} \sqrt{n} p_i.$$

Here we used the inequalities (1), (2) and (3). If we furthermore apply the straightforward relations $p_i \leq F_i$ for $1 \leq i \leq n$ and $\sum_{i=1}^{n} F_i = F^*_\text{pmtn}$ together with $|L| \leq \sqrt{n}$, then we derive that

$$F^H \leq F^*_\text{pmtn} + \frac{3}{2} \sqrt{n} \sum_{i \in S \cup L \cup F} F_i + |L| \sum_{j=1}^{n} p_j \leq \left(1 + \frac{5}{2} \sqrt{n}\right) F^*_\text{pmtn}$$

holds. Since $F^*_\text{pmtn} \leq F^*$, this proves that the algorithm has a worst-case performance guarantee of $O(\sqrt{n})$.

What about the time complexity? The SRPT-schedule can be computed in $O(n \log n)$ time. Procedures SmallSubtree and FatherSon always consider only $O(\sqrt{n})$ of the jobs, whereas LastRoot may have to deal with up to $O(n)$ jobs. Summarizing, this yields

$$|S| \cdot O(\sqrt{n}) + |L| \cdot O(n) + |F| \cdot O(\sqrt{n}) \leq O(n^{3/2})$$

accesses and changes to the jobs. If the jobs are stored in a binary search tree, every modification of a job can be done in $O(\log n)$ time. Hence, the overall complexity of the algorithm is $O(n^{3/2} \log n)$.

To see that the algorithm can be a factor of $\Omega(\sqrt{n})$ away from the optimum, consider the following instance with $n = x^2$ jobs: The jobs $J_i$ with $1 \leq i \leq x - 1$ all have processing time $1/(x-1)$ and release times $1 + (i-1)/(x-1)$. Job $J_x$ has processing time $x^2$ and release time 0. The remaining $x^2 - x$ jobs are dummy jobs, all with processing times 0 and release times 0. The forest corresponding to the optimum preemptive schedule has a single non-trivial tree with root $J_x$ and with $J_1, \ldots, J_{x-1}$ as leaves. Hence, our algorithm calls SmallSubtree($x$) and produces a schedule with $F^H = x^3 - x + 1$, whereas $F^* = x^2 + 3$ holds. As $x$ tends to infinity, the ratio $F^H/F^*$ grows like $\Theta(x) = \Theta(\sqrt{n})$. 

---

Figure 4: Illustration for the lower bound example in Corollary 2.4.
Corollary 2.4. For any instance of $n$ jobs with release times on a single machine, the inequality $F^{*}/F^{\text{prtn}} \leq \frac{5}{2}\sqrt{n} + 1$ holds. Moreover, for every sufficiently large $n$, there exists an instance $I_n$ such that $F^{*}(I_n)/F^{\text{prtn}}(I_n) \approx \sqrt{n}$.

Proof. The upper bound follows from inequality (4) and $F^{*} \leq F^H$. The lower bound follows from the instance with $n$ jobs where $r_1 = 0$, $p_i = n$, $r_i = (i-1)/\sqrt{n}$, and $p_i = \varepsilon = 1/n$ for $i \geq 2$. Then $F^{\text{prtn}} \approx n$ and $F^{*} \approx n^{3/2}$ holds (see Figure 4 for an illustration).

3 The Non-Approximability Result

In this section, we will prove that no polynomial time approximation algorithm for minimizing the total flow time may have worst-case performance guarantee $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$. The proof is done by a reduction from the following version of the NP-complete Numerical 3-Dimensional Matching problem (see Garey and Johnson [11]).

Problem. Numerical 3-Dimensional Matching (N3DM)

Instance. Positive integers $a_i, b_i$ and $c_i$, $1 \leq i \leq k$, with $\sum_{i=1}^{k}(a_i + b_i + c_i) = kD$.

Question. Do there exist permutations $\pi, \psi$ such that $a_i + b_{\pi(i)} + c_{\psi(i)} = D$ holds for all $i$?

This problem remains NP-complete even if the numbers $a_i$, $b_i$ and $c_i$ are encoded in unary and the total size of the input is $\Theta(kD)$. In the following, we will make use of this unary encoding. Consider an arbitrary instance of Numerical 3-Dimensional Matching and let $0 < \varepsilon < \frac{1}{2}$ be some real number. Define numbers

$$n = \lceil (20k)^{4/\varepsilon} D^{2/\varepsilon} \rceil, \quad r = \lceil 2 Dn^{(1-\varepsilon)/2} \rceil, \quad g = 100r k^2.$$

Next we will construct from the N3DM instance and from the number $\varepsilon$, a corresponding scheduling instance with $n$ jobs. For every number $a_i$ in the N3DM instance, we introduce a corresponding job with processing time $2r + a_i$, for every $b_i$ we introduce a job with processing time $4r + b_i$ and for every $c_i$ we introduce a job with processing time $8r + c_i$. These $3k$ jobs are called the big jobs and they are all released at time $0$.

Moreover, there will be a number of so-called tiny jobs. Tiny jobs only occur in groups denoted by $G(t; \ell)$, where $t$ and $\ell$ are positive integers. A group $G(t; \ell)$ consists of $\ell$ tiny jobs with processing time $1/\ell$. They are released at the times $t + i/\ell$ for $i = 0, \ldots, \ell - 1$. Note that it is possible to process all jobs in $G(t; \ell)$ in a feasible way during the time interval $[t, t+1]$ with a total flow time of $1$. We introduce the following groups of tiny jobs.

(T1) For $1 \leq i \leq k$, we introduce the group $G((14r + D + 1)i - 1; rg)$.

(T2) For $1 \leq i \leq g$, we introduce the group $G((14r + D + 1)k + ri - 1; g)$.

In a pictorial setting, the groups of type (T1) occur at regular intervals from time $14r + D$ to time $(14r + D + 1)k$. They leave $k$ holes, where each hole has length $14r + D$. Similarly, the groups of type (T2) occur in a regular pattern after time $(14r + D + 1)k$. They leave holes of size $r - 1$.

So far we have introduced $3k$ big jobs and $k + 100rk^2$ groups of tiny jobs, which amounts to an overall number of

$$3k + krg + 100rk^2g = 3k + 100r^2k^3 + 10,000r^2k^4 < 11,000r^2k^4 < n$$
jobs. In order to simplify calculations, we introduce a number of artificial jobs all with processing time zero and release time zero such that the total number of jobs becomes equal to \( n \). This completes the construction of the scheduling instance.

**Lemma 3.1** If the N3DM instance has a solution, then for the constructed scheduling instance \( F^* \leq 200rk^2 \) holds.

**Proof.** Consider the following feasible schedule. All tiny jobs are processed immediately at their release times. Hence, their total flow time equals \( k + 100rk^2 \). For every triple \((a_i, b_{\sigma(i)}, c_{\psi(i)})\) with sum \( D \) in the solution of the N3DM instance, we pack the corresponding three jobs together into one of the holes of length \( 14r + D \) that are left free by the groups of type \((T1)\). The job corresponding to \( a_i \) is processed first, the job corresponding to \( b_{\sigma(i)} \) is processed second and the job corresponding to \( c_{\psi(i)} \) is processed last in this hole. It is easy to see that this yields a total flow time of

\[
k + 100rk^2 + 3 \sum_{i=1}^{k} (a_i + 2r) + 2 \sum_{i=1}^{k} (b_i + 4r) + \sum_{i=1}^{k} (c_i + 8r) + \frac{3}{2}(14r + D + 1)k(k - 1).
\]

Since \( \sum_{i=1}^{k} (a_i + b_i + c_i) = kD \) and since \( D \leq r \), one easily gets an upper bound of

\[
k + 100rk^2 + 25rk + 24rk(k + 1) < 200rk^2
\]

for the expression in (5).

**Lemma 3.2** If the N3DM instance does not have a solution, then for the constructed scheduling instance \( F^* \geq 100r^2k^2 \) holds.

**Proof.** Consider an optimum schedule \( S^* \) and suppose that its total flow time is strictly less than \( 100r^2k^2 \). Our first claim is that in \( S^* \) all tiny jobs in groups of type \((T1)\) are processed as soon as they are released: Since all these tiny jobs have identical processing times, they are processed in \( S^* \) in order of increasing release times. It is easy to see that there is no use in splitting one of the groups of type \((T1)\) by processing some larger jobs in between (since this would contradict the SPT rule). Since all big jobs have integer processing times and since the total processing time in every group is equal to one, we may further assume that the starting time of every big job and of every group \( G(t; \ell) \) in \( S^* \) is integer. In case that the processing of some group \( G(t; r) \) starts at time \( t + x \), for \( x \) an integer, the total flow time of the \( rg \) tiny jobs in \( G(t; r) \) is \( rgx + 1 = 100r^2k^2x + 1 \). This implies \( x = 0 \) and proves the claim.

Our second claim is that in \( S^* \) none of the big jobs is processed during the time interval that starts with release of the last group of type \((T1)\) and ends with release of the last group of type \((T2)\). Suppose otherwise. The groups of type \((T2)\) are released at regular intervals of length \( r \). Since big jobs all have processing times of at least \( 2r \), scheduling a big job somewhere between these groups would shift the jobs of at least one group by at least \( r \) time units away from their release time. This would yield a total flow time of at least \( gr = 100r^2k^2 \) and proves the second claim.

Our third claim is that in \( S^* \) one of the big jobs is processed after the last group of tiny jobs. Suppose otherwise. There are two types of holes that are left free by the tiny jobs for processing the big jobs: \( k \) holes of length \( 14r + D \) and \( 100rk^2 \) holes of size \( r - 1 \). Since big jobs all have
processing times of at least $2r$, they must be packed into the holes of size $14r + D \leq 15r$. Since two jobs corresponding to numbers $c_i$ and $c_j$ have total processing time at least $16r$, every such hole must contain exactly one job corresponding to some $c_i$. By analogous arguments, we get that every hole of size $14r + D$ must contain exactly one job corresponding to some $a_i$, $b_j$, and $c_k$, respectively. This implies that the corresponding three numbers fulfill $a_i + b_j + c_k \leq D$, which in turn yields that $a_i + b_j + c_k = D$ (since the total sum of these $3k$ numbers is $kD$ and the total size of the holes is $kD$). Hence, the N3DM instance would have solution; a contradiction.

Finally observe that the last group of tiny jobs is processed at time $t = (14r + D + 1)k + 100r^2k^2 - 1$. Hence, the big job that is processed after this last group has a completion time of at least $100r^2k^2$, and since its release time is zero, its now time is at least $100r^2k^2$. ■

**Theorem 3.3** For all $0 < \varepsilon < \frac{1}{2}$ and $0 < \alpha < 1$, there does not exist a polynomial time approximation algorithm for the minimum total flow time problem with worst-case approximation guarantee $o(n^{\frac{1}{2}-\varepsilon})$ unless $P = NP$.

**Proof.** Suppose such an approximation algorithm $H$ would exist for some fixed $0 < \varepsilon < \frac{1}{2}$ and $0 < \alpha < 1$. Take an instance of N3DM that is encoded in unary, and perform the above construction. Since the instance is encoded in unary, the size of the resulting scheduling instance is polynomial in the size of the N3DM instance.

Then apply algorithm $H$ to the scheduling instance. In case the N3DM instance has a solution, Lemma 3.1 and the worst-case performance guarantee of $H$ imply that

$$F^H \leq \alpha n^{\frac{1}{2}-\varepsilon} \cdot F^* < \frac{n}{2} \cdot 200r^2k^2 = 100r^2k^2.$$ 

In case the N3DM instance does not have a solution, Lemma 3.2 implies that

$$F^H \geq F^* \geq 100r^2k^2$$

holds. Hence, with the help of algorithm $H$ we could separate between these two possibilities in polynomial time, which would imply $P = NP$. ■

4 Concluding Remarks

In this paper we investigated the problem of minimizing total flow time on a single machine. For this NP-complete problem, only approximation algorithms with worst-case bounds of $\Omega(n)$ were known up to now. We presented the first approximation algorithm with sublinear worst-case performance guarantee. The algorithm has a tight worst-case of $O(\sqrt{n})$. It is based on a resolution of the interrupted jobs in the corresponding optimum preemptive schedule. Moreover, we derived a lower bound on the worst-case performance guarantee of polynomial time approximation algorithms for this problem. We proved that no polynomial time algorithm can have a worst-case performance guarantee of $O(n^{\frac{1}{2}-\varepsilon})$ with $\varepsilon > 0$. This also shows that our approximation algorithm is almost ‘best possible’.

We finish the paper with some remarks and open problems.

(1) The open main problem concerns closing the gap between our upper bound and our lower bound. E.g. there might exist an approximation algorithm with worst-case performance guarantee of $O(\sqrt{n/\log n})$. 


(2) Our lower bound argument carries over to the problem \( Pm|r_i| \sum F_i \) with parallel machines, but our algorithm breaks down for this case: For \( m \geq 2 \) machines the preemptive problem \( Pm|r_i| \sum F_i \) is NP-complete (Du, Leung and Young [9]) and so the algorithm does not have the optimum preemptive schedule as a starting point.

(3) Another open problem is designing approximation algorithms for the total weighted flow time problem on a single machine. Also in this case, the corresponding preemptive problem \( 1|pmtn, r_i| \sum w_i F_i \) is NP-complete (Labetoulle, Lawler, Lenstra and Rinnooy Kan [14]).

(4) In some schedule, the waiting time \( W_i \) of some job \( J_i \) is defined by \( W_i = S_i - r_i \), i.e. the time the job spends in the system while waiting for being processed. We are not aware of any positive results on approximating this problem.

Acknowledgement. Gerhard Woeginger acknowledges helpful discussions with Amos Fiat and Stefano Leonardi.

References