Minimum Makespan Scheduling

Approximation Algorithms 2009

Petros Potikas

Minimum makespan scheduling

Definition**:** Let $p_1, p_2, \ldots, p_n$ be the processing times for n jobs and m identical machines.

Goal**:** Find an assignment of the n jobs to the m machines, so that the completion time, also called *makespan*, is minimized.

Minimum makespan scheduling

Results

- Strongly NP-hard problem
- Approximation algorithm with ratio 2
- PTAS
- No FPTAS

Minimum makespan scheduling

Lower bounds

1. The average time for which a machine has to run, $(\sum_i p_i)/m$,
2. The last processing time.

$$\text{LB} = \max\{(\sum_i p_i)/m,\ max_i\{p_i\}\ \}$$

Minimum makespan scheduling

Algorithm 1 (Graham, 1966)

1. Order the n jobs arbitrarily.

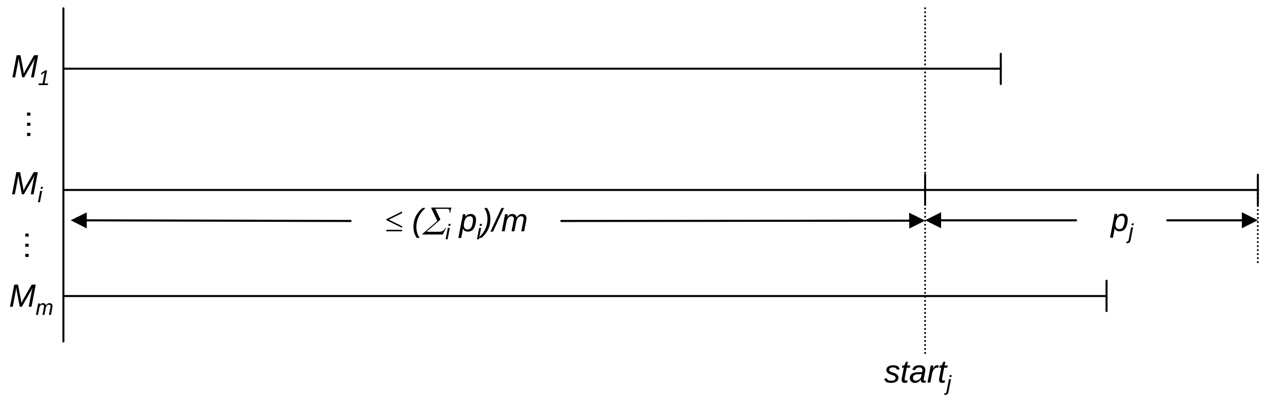
2. Schedule jobs on machines in this order, scheduling the next job on machine that has been assigned least so far.

Minimum makespan scheduling

Theorem 1: *Algorithm 1 achieves a 2-approximation.*

Proof:

Let M_i be the machine that completes last in the schedule produced by the algorithm and let j be the last job scheduled on this machine.



Let $start_j$ be the time that job j starts.
From the choice of M_i by the algorithm we know that

all the other machines are busy until $start_j$

Thus, $start_j \leq (\sum_i p_i)/m \leq \mathrm{OPT}$

Minimum makespan scheduling

Theorem 1: *Algorithm 1 achieves an approximation factor* 2.

Proof (cont'd):
Furthermore, $p_j \leq \text{OPT}$

Thus, the makespan produced by the algorithm is

$$start_j + p_j \leq 2 \cdot \text{OPT}$$

We also proved, that $\text{LB} \leq \text{OPT} \leq 2 \cdot \text{LB}$. □

Minimum makespan scheduling

Tight example:

A sequence of m^2 jobs with unit processing time, followed by a single job of length m.

OPT = $m+1$, while the algorithm gives makespan $2m$.

Minimum makespan scheduling

Algorithm 2 (Graham)

1. Sort the n jobs by decreasing processing times.

2. Schedule jobs on machines in this order, scheduling the next job on machine that has been assigned least so far.

Theorem 2: *Algorithm 2 achieves a* 4/3-*approximation*.

Tight example:
m machines, n=2m+1 jobs
two jobs of length m+1, m+2,…, 2m
one job of length m

A PTAS for minimum makespan scheduling

We will, for every ε>0, derive an algorithm A_ε that

- Returns a schedule with makespan $\leq (1+3\varepsilon)\mathrm{OPT}$
- Runs in time $O(n^{2k \lceil log_2(1/\varepsilon) \rceil})$ where $k = \lceil log_{1+\varepsilon}(1/\varepsilon) \rceil$

A_ε is therefore a

Polynomial Time Approximation Scheme (PTAS)

but not a

Fully Polynomial Time Approximation Scheme (FPTAS)

(in an FPTAS, time is not only polynomial in n but also in $1/\varepsilon$)

Restricted bin packing

There a exists a schedule with makespan t iff n objects of sizes $p_1, p_2, \ldots, p_n$ can be packed into m bins of capacity t.

Reduction from mininum makespan to bin packing:
Let I be the sizes of the n objects, $p_1, p_2, \ldots, p_n$ and bins(I,t) the minimum number of bins of size required to pack these n objects.

$$\text{OPT(makespan)} = \min\{t : \text{bins}(I,t) \leq m\}$$

We know that

$$LB \leq t \leq 2 \cdot LB$$

So the idea is to binary search $[LB, 2 \cdot LB]$ to find the minimum t for which $\text{bins}(I,t) \leq m$.

We can't do this exactly!

Core algorithm: restricted bin packing (fixed number of object sizes), of time $O(n^{2k})$ that uses $\alpha(I,t,\varepsilon)$ bins of size $t(1+\varepsilon)$.

This packing has the property

$$\forall t,\varepsilon \qquad \alpha(I,t,\varepsilon) \leq \mathrm{bins}(I,t)$$

Thus $\qquad \forall \varepsilon \qquad \alpha(I,2\mathrm{LB},\varepsilon) \leq \mathrm{bins}(I,2\mathrm{LB}) \leq m$

So, the PTAS is the following:

- If $\alpha(I,\mathrm{LB},\varepsilon) \leq m$ then use packing given by core algorithm for t=LB. This has makespan
$$\leq \mathrm{LB}(1+\varepsilon) \leq \mathrm{OPT}(1+\varepsilon)$$

- If $\alpha(I,\mathrm{LB},\varepsilon) > m$, then perform a binary search to find an interval $[T',T]$ in [LB,2LB] with $T-T' \leq \varepsilon\mathrm{LB}$, $\alpha(I,T',\varepsilon) > m$ and $\alpha(I,T,\varepsilon) \leq m$. Return the packing given by the core algorithm for $t=T$.

Notice that $m < \alpha(I,T',\varepsilon) \leq \mathrm{bins}(I,T')$, so $T' \leq \mathrm{OPT}$ and

$$T \leq T' + \varepsilon\mathrm{LB} \leq \mathrm{OPT} + \varepsilon\mathrm{OPT} \leq (1+\varepsilon)\mathrm{OPT}$$

The *core* algorithm for $t=T$ returns a schedule (packing) with makespan at most $(1+\varepsilon)T$. The makespan of the schedule returned is at most

$$(1+\varepsilon)T \leq (1+\varepsilon)^2\text{OPT} \leq (1+3\varepsilon)\text{OPT}$$

The binary search uses at most $log_2 1/\varepsilon$ steps.

Error introduced by two sources:

- o Rounding objects so that there a bounded number of different sizes
- o Terminating the binary search to ensure polynomial running time

Exact restricted bin packing

n items to pack in bins of size t, with k different sizes only

Input $I=(i_1,i_2,\ldots,i_k)$ (fix an ordering on the object sizes)

BINS$(i_1,i_2,\ldots,i_k)$: minimum number of bins needed to pack these objects

Suppose we are given $(n_1,n_2,\ldots,n_k)$, $\sum_i n_i=n$

First, compute Q, the set of all k-tuples $(q_1,q_2,\ldots,q_k)$, such that BINS$(q_1,q_2,\ldots,q_k)=1$ (at most $O(n^k)$ such tuples)

Exact restricted bin packing

Use dynamic programming to find all the entries of the table $\text{BINS}(i_1,i_2,\ldots,i_k)$, for $0 \leq i_j \leq n_j$

1. $\forall q \in Q$ set $\text{BINS}(q) = 1$
2. If $\exists$j, such that $i_j < 0$ then set $\text{BINS}(i_1,i_2,\ldots,i_k) = \infty$
3. For all other q, use recurrence relation
   $\text{BINS}(i_1,i_2,\ldots,i_k) = 1 + \min_{(q1,q2,\ldots,qk) \in Q} \text{BINS}(i_1\text{-}q_1,i_2\text{-}q_2,\ldots,i_k\text{-}q_k)$

Since there are $O(n^k)$ entries and each one takes $O(n^k)$ time, the algorithm needs $O(n^{2k})$ time.

The Core Algorithm

$t \in [\mathrm{LB}, 2\mathrm{LB}]$, so $\forall j, p_j \leq t$

1. An object is small if it has size $\leq t\varepsilon$.
2. Non-small objects are *rounded*.

If $p_j \in [t\varepsilon(1+\varepsilon)^i, t\varepsilon(1+\varepsilon)^{i+1}]$, then set $p_j{}' = t\varepsilon(1+\varepsilon)^i$. There can be at most $k = \lceil \log_{1+\varepsilon} 1/\varepsilon \rceil$ different sizes.

3. Use dynamic programming algorithm to optimally pack non-small objects using $p_j{}'$ costs into bins of size t.

   Rounding can reduce size by a factor of $1+\varepsilon$ at most, so packing is valid for bins of size $t(1+\varepsilon)$ with the original p_j object sizes.

4. Place the small objects items into the $t(1+\varepsilon)$ packing greedily. Open new bins only if needed. If new bins are opened, then all other must be filled at height t at least.
5. Let $\alpha(I,t,\varepsilon)$ be the number of bins used (of size $t(1+\varepsilon)$).

The Core Algorithm

Lemma: $\alpha(I,t,\varepsilon) \leq \text{bins}(I,t)$.

Proof:

Case 1: The algorithm opens new bins. Then all the bins, except possibly the last one, are filled to at least size t. Thus, the optimal packing into bins of size t must use at least $\alpha(I,t,\varepsilon)$ bins.

Case 2: The algorithm does not open new bins. Let I' be the set of non-small items. Then

$$\begin{aligned}\alpha(I,t,\varepsilon) &= \alpha(I',t,\varepsilon)\\ &\leq \text{bins}(I',t)\\ &\leq \text{bins}(I,t).\end{aligned}$$

The optimal packing of I' uses $\text{bins}(I',t)$ bins, so the same packing of the rounded down I' also uses $\text{bins}(I',t)$ bins.

But $\alpha(I',t,\varepsilon)$ is the *optimal* number of bins needed for the rounded down I'.

The first inequality holds.

Packing optimally more items can not reduce the number of bins needed.□