CMPUT 675: Approximation Algorithms **Fall 2014**

Lecture 7 (Sep 17): PTAS & Min-Makespan on Identical Parallel Machines

Lecturer: Zachary Friggstad *Scribe: Chris Martin*

7.1 PTAS & FPTAS

Last lecture, we saw a $(1-\epsilon)$-approximation for the KNAPSACK problem with running time $O(n^3/\epsilon)$. There is a special name for approximations that can get within any constant factor of the optimum in polynomial time.

Definition 1 *A **polynomial-time approximation scheme** (or **PTAS**) is an approximation algorithm that takes an additional parameter $\epsilon > 0$, and finds a $(1 \pm \epsilon)$-approximate solution. The running time must be polynomial for constant ϵ (this includes algorithms with running times like $n^{O(1/\epsilon)}$ or even $n^{O(2^{1/\epsilon})}$).*

Definition 2 *A **fully polynomial-time approximation scheme** (or **FPTAS**) is a PTAS with running time polynomial in both n and ϵ. This includes algorithms with running time like $O(n^3/\epsilon)$, for example, but not $n^{O(1/\epsilon)}$.*

The algorithm for KNAPSACK we saw last lecture was an FPTAS. This lecture, we will see a PTAS for a new problem. In the next assignment, you will prove there is no FPTAS for this problem unless P = NP.

7.2 Minimizing Makespan on Identical Parallel Machines

Consider a set of jobs $J = \{1, 2, \ldots n\}$ where each job needs integer processing time $p_j \geq 0$. Furthermore, we have a set of identical machines $K = \{1, 2, \ldots k\}$ running in parallel. Assign a set of jobs to each machine such that the maximum time any machine takes to complete its set of jobs (or its *makespan*) is minimized. More formally, find $\Phi : J \to K$ to minimize

$$\max_{1 \leq i \leq k} \sum_{j \in \Phi^{-1}(i)} p_j.$$

We say that the *load* of a machine in a given solution is the running time of all jobs assigned to that machine.

7.2.1 A Simple 2-Approximation

Algorithm 1 Simple Approximation

1: **for** all $j \in J$ **do**
2: Assign j to the machine with least load so far.
3: **end for**

Theorem 1 *Algorithm 1 is a 2-approximation.*

Proof. There are two crucial observations to be made:

1. $p_j \leq OPT$, and
2. $\sum_{j=1}^{n} \frac{p_j}{k} \leq OPT$ (that is, the average machine load must be $\leq OPT$).

Say job j was assigned to machine i and was the *last* job to terminate overall in the generated schedule. The load of i before being assigned j by the algorithm was then at most $\sum_{j=1}^{n} \frac{p_j}{k} \leq OPT$, since i was the machine with minimum load at that step. Adding job j then increases the load to at most $OPT + p_j \leq 2 \cdot OPT$, which is exactly the approximation ratio of the schedule since j is the last job. ■

This approximation appears in [G66]. The Williamson and Shmoys text shows that this greedy algorithm is in fact a 4/3-approximation if the loop considers jobs in decreasing order of processing time.

7.2.2 A PTAS-Approximation

Theorem 2 *There is a PTAS for* Minimizing Makespan on Identical Parallel Machines

The algorithm uses the following as a subroutine.

Theorem 3 *Given a value $T \geq 0$, there is an $n^{O(1/\epsilon^2)}$-time algorithm that either:*

1. *Returns a solution with makespan at most $(1+\epsilon) \cdot \max(T, OPT)$, or*
2. *Determines there is no solution with makespan $< T$.*

Furthermore, if $T \geq OPT$ then the algorithm is guaranteed to find a solution with makespan at most $(1+\epsilon) \cdot \max(T, OPT)$.

For now, assume Theorem 3 holds.

Proof of Theorem 2. Using the algorithm in Theorem 3, a binary search can be performed to find the smallest T such that a solution with makespan at most $(1+\epsilon) \cdot \max(T, OPT)$ exists. Let P denote $\sum_{j=1}^{n} p_j$, the total running time of all jobs. We know $0 \leq OPT \leq P$ and that OPT is an integer (since all p_j are integers) so the number of calls to the algorithm in Theorem 3 is $O(\log P)$.

This T is at most OPT, so the makespan is at most $(1+\epsilon) \cdot OPT$. The running time of this algorithm is $O(n^{O(1/\epsilon^2)} \cdot \log P)$ which is polynomial in the size of the input for constant values ϵ (at least $\log_2 P$ bits of the input are used just to represent the running times p_j). ■

All that is left is to prove Theorem 3. The next claim simplifies things and shows we only have to focus on scheduling the jobs with somewhat large processing time. For the rest of this discussion, let $J_{small} = \{j \in J : p_j \leq \epsilon \cdot T\}$ and $J_{large} = J - J_{small}$.

Claim 1 *Given a solution of makespan $(1+\epsilon) \cdot T$ using only jobs in J_{large}, greedily placing the jobs in J_{small} (as in Algorithm 1) results in makespan at most $(1+\epsilon) \cdot \max(T, OPT)$.*

Proof of Claim 1. Say machine i has the highest load. If it has no jobs in J_{small} assigned to it, then its load is at most $(1+\epsilon)\cdot T$. Otherwise, say $\bar{j} \in J_{small}$ was added last. The load is then at most

$$\begin{aligned} p_{\bar{j}} + \frac{1}{k}\sum_{j=1}^{n} p_j &\leq \epsilon \cdot T + OPT \\ &\leq (1+\epsilon)\cdot \max(T, OPT). \end{aligned}$$

■

Proof of Theorem 3. If $p_j > T$ for some j, then clearly there is no solution with makespan at most T. So, assume all processing times are at most T.

We will use dynamic programming to find a solution with makespan $(1+\epsilon)\cdot T$ over J_{large} (or else determine that makespan $\leq T$ solution exists).

Let b be the smallest integer such that $1/b \leq \epsilon$. Note that for $\epsilon \leq 1$ we have $b \geq 2/\epsilon$ (and if $\epsilon > 1$ we may as well just use the 2-approximation from Section 7.2.1).

Define new processing times $p'_j = \lfloor \frac{p_j b^2}{T} \rfloor \cdot \frac{T}{b^2}$ (i.e. scale the p_j values to integer multiples of $\frac{T}{b^2}$). It is then the case that $p'_j \leq p_j \leq p'_j + \frac{T}{b^2}$, and further, $p'_j = m \cdot \frac{T}{b^2}$ for some $m \in \{b, b+1, \ldots b^2\}$ ($m \geq b$ since $p_j \geq \frac{T}{b}$ by the definition of J_{large}).

For the DP, define a *configuration* as a tuple $(a_b, a_{b+1}, \ldots a_{b^2})$ of nonnegative integers such that $\sum_{i=b}^{b^2} a_i \cdot i \cdot \frac{T}{b^2} \leq T$. In such a configuration, a_i represents the number of jobs with running time $i \cdot T/b^2$. Let $\mathcal{C}(T)$ be the set of all configurations. Thus, there is a clear correspondence between configurations and assignments of jobs to a given machine with makespan (under processing times p') at most T.

The DP table is defined as follows. Given integers $n_b, n_{b+1}, \ldots n_{b^2} \geq 0$ where n_i indicates the number of jobs with processing time $i \cdot \frac{T}{b^2}$ that need to be run, let $f(n_b, n_{b+1}, \ldots n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan at most T. Stated as a recurrence,

$$\begin{aligned} f(0,0,\ldots 0) &= 0 \\ f(n_b, n_{b+1}, \ldots n_{b^2}) &= 1 + \min_{\{(a_b,\ldots a_{b^2}) \in \mathcal{C}(T) : \forall i, a_i \leq n_i\}} f(n_b - a_b, \ldots n_{b^2} - a_{b^2}) \end{aligned}$$

Note that $\forall i, n_i \leq n$, so the number of table entries and unique configurations $(a_b, \ldots a_{b^2})$ in the DP are both bounded by n^{b^2}. Thus, filling the DP table takes at most $n^{O(b^2)}$ time.

Using this DP, if $\overline{n_b}, \ldots \overline{n_{b^2}}$ is the original configuration tuple for all jobs $\in J_{large}$, then if $f(\overline{n_b}, \ldots \overline{n_{b^2}}) \leq k$, output **yes** (the actual assignment can be constructed using standard techniques for recovering a solution from a DP table). Otherwise output **no**. Note that each machine is assigned at most b jobs since $p'_j \geq \frac{T}{b}$ for $\forall j \in J_{large}$, and the p'-makespan is $\leq T$. Therefore, the true makespan is

$$\begin{aligned} &\leq p'\text{-makespan} + b \cdot \max_{j \in J_{large}} (p_j - p'_j) \\ &\leq T + b \cdot \frac{T}{b^2} \\ &= (1+\epsilon)\cdot T. \end{aligned}$$

■

This PTAS appeared in [HS87].

References

G66 R. L. Graham, Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

HS87 D. S. Hochbaum and D. B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results, *Journal of the ACM*, 34:144–162, 1987.