A PTAS for the \textsc{Makespan} Problem

The Class \textit{\textsc{SNP}}

\textbf{Definition 1} A problem \( \mathcal{A} \) is said to be \textit{Strongly NP-hard} or \textit{\textsc{SNP-hard}} if every problem in \( \mathcal{NP} \) can be reduced to \( \mathcal{A} \) in polynomial time, such that the numbers (values specified in the problem instance) are expressed in unary form in the reduced problem.

This definition implies that an \textit{\textsc{SNP}}-hard problem cannot have a pseudo-polynomial time algorithm, unless \( \mathcal{P} = \mathcal{NP} \). Most known \( \mathcal{NP} \) - hard problems are \textit{\textsc{SNP}}-hard, ex. \textsc{Set-Cover}, \textsc{TSP}. The \textsc{Knapsack} problem is an example of a non \textit{\textsc{SNP}}-hard problem, and we saw a pseudo-poly time algorithm for it in Lecture 5. The set of problems with no pseudo-poly time algorithm is a proper subset of the the class \textit{\textsc{SNP}}. It may be shown, under certain weak restrictions (and the assumption \( \mathcal{P} \neq \mathcal{NP} \)), that a problem which admits an FPTAS also admits a pseudo-poly time algorithm. This in turn implies that \textit{\textsc{SNP}}-hard problems do not admit an FPTAS.

\textbf{Theorem 1} The minimum \textsc{makespan} problem is \textit{\textsc{SNP}}-hard. Hence, it does not admit an FPTAS, unless \( \mathcal{P} = \mathcal{NP} \).

In this lecture, we show the following result.

\textbf{Theorem 2} There exists a \((1+\epsilon)\) approximation algorithm for minimum \textsc{makespan} with time complexity \( O(n^{2/\epsilon^2}) \).

Note that this is polynomial time only for a constant \( \epsilon \) and hence, does not satisfy the conditions for a FPTAS.

\footnotesize
\begin{itemize}
\item \textsuperscript{1}Dept of Comp. Sci., Stanford Univ.
\item \textsuperscript{2}Dept of Comp. Sci. and Automation, Indian Institute of Science, Bangalore.
\item siddu@csa.iisc.ernet.in
\end{itemize}

1
Definition 2 A decision procedure is an algorithm which takes as input an instance of a minimisation (maximisation) problem \( I \) and a target value \( T \) and returns either Yes if there exists a solution of value \( < T \) (\( > T \)) or No if no such solution exists.

Theorem 3 If there exists a polynomial time decision procedure for Makespan, then there exists a poly-time approximation algorithm for Makespan.

Sketch of Proof The optimal value of Makespan \( C^* \) satisfies \( 0 < C^* < \sum p_i \leq np_{\text{max}} \), where \( p_i \)'s are the processing times, \( n \) is the number of jobs, and \( P_{\text{max}} \) is the max processing time. Since the optimal solution is bracketed in the above interval, perform a search for the solution using the decision procedure. The search takes at most \( \log np_{\text{max}} \) (log in the size of the interval, if the search is binary) and since each step is poly-time, this gives us the required poly-time algorithm. Note that here we are assuming a decision procedure which returns the actual solution, in addition to the Yes/No answers.

From the non-approximability result of Theorem 1 we know that such a decision procedure (one that leads to a fully polynomial time algorithm) cannot exist. However, we next show a decision procedure on a subclass of the Makespan problems and use it to obtain the PTAS.

A Restricted Makespan Problem

Definition 3 Let Makespan \( (S) \) denote an instance of the Makespan problem, with the processing times restricted to be from a finite number of possible values, \( p_i \in \{z_1, z_2, \ldots, z_s\}; \forall i. \)

The problem is then specified by specifying a list of values for processing times, and for each value, the number of jobs having that processing time. The inputs to the decision procedure are

1. The target value \( T \),
2. An ordered set of values of the processing time \( Z = \{z_1, z_2, \ldots, z_s\} \),
3. The number of jobs corresponding to each processing time \( N = \{n_1, n_2, \ldots, n_s\} \) and
4. The number of machines \( m \).

Theorem 4 Makespan \( (S) \) has time complexity \( O(n^{2s}) \), where \( n = \max_i n_i \).
A Dynamic Programming Approach to \textsc{Makespan}(S)

Consider the problem of scheduling jobs with processing times \( \in \mathbb{Z} \) on a single machine. There are \( x_i \) jobs of size \( z_i \), for each \( i \), and the total load should not exceed \( T \). i.e. Find \( v_i \)'s satisfying

\[
\sum_i v_i z_i \leq T \\
v_i \leq x_i \quad \forall i
\]

Let \( V(x_1, x_2, \ldots, x_s) \) denote the feasible solution space for this problem:

\[
V(x_1, x_2, \ldots, x_s) = \left\{ (v_1, v_2, \ldots, v_s) \mid \sum_i v_i z_i \leq T, v_i \leq x_i \right\}.
\]

The size of this set is \( O(n^s) \) and for a constant \( s \), a brute force search of \( V \) to find the best schedule for a single machine (i.e. one with maximum load \( < T \)) is still poly-time. For the general problem with multiple machines, let \( M(x_1, x_2, \ldots, x_s) \) denote the minimum number of machines required to schedule \( x_i \) jobs of size \( z_i \), such that the load on each machine is at most \( T \). Note that the solution to \textsc{Makespan}(S) is just \( \min_{M(n_1, n_2, \ldots, n_s) \leq mT} \).

To frame a dynamic programming problem, see that the solution to the single machine problem can be used to solve the multi machine problem by fixing the best solution \((v_1, v_2, \ldots, v_s) \in V \) for one machine and recursively finding the best solution for the remaining jobs on the remaining machines.

\[
M(x_1, x_2, \ldots, x_s) = 1 + \min_M M(x_1 - v_1, \ldots, x_s - v_s).
\]

The table for \( M(n_1, n_2, \ldots, n_s) \) is of size \( n^s \) and the computation of each entry requires enumerating \( V(x_1, x_2, \ldots, x_s) \) which is again \( O(n^s) \). Hence the dynamic program and the decision process has complexity \( O(n^{2s}) \) which is polynomial if \( s \) is a constant (note that this is where the restriction comes into play). A binary search using this decision procedure yields an algorithm for \textsc{Makespan}(S). Call this algorithm \textsc{DynMakespan}.S.

Relaxed Decision Procedures

\textbf{Definition 4} A \( \rho \)-relaxed decision procedure where \( \rho > 1 \) takes as input a problem \( P \) and a target value \( T \) and returns:

1. No, if there exists no solution to \( P \) of value less than \( T \).
2. Yes, if there exists a solution \( C \leq \rho T \).
Figure 1: Output of a $\rho$-relaxed decision procedure for \textsc{Makespan}(S)

Note that if the optimal solution is $C^*$ and for a $T \in [C^*/\rho, C^*)$, the relaxed decision procedure may return a No, even though there exists a solution $\leq \rho T$. This is illustrated in Figure 1.

\textbf{Theorem 5} A $\rho$ relaxed decision procedure for \textsc{Makespan}(S) yields a $\rho$-approximation algorithm.

Use the decision procedure in a binary search in the (bounded) interval. The minimum $T$ beyond which the $\rho$-relaxed decision procedure returns a No is arbitrarily close to a $\rho$ approximation, but number of steps in the search will depend on the accuracy desired. We next show that a relaxed decision procedure exists for \textsc{Makespan}, thereby establishing Theorem 2.

\textbf{Theorem 6} There exists a $(1 + \epsilon)$-relaxed decision procedure for \textsc{Makespan}(S) of time complexity $O(n^{2/\epsilon^2})$.

\textbf{The PTAS for Makespan}

We solve the $(1 + \epsilon)$-relaxed decision procedure for \textsc{Makespan} by first solving it for the restricted class of problems, and then showing how the general problem can be solved using the solution to the restricted problem.

\textbf{Theorem 7} For \textsc{Makespan}(S) problems with $\rho \geq \epsilon T$, $\forall j$, then there exists an $O(n^{2/\epsilon^2})$ time $(1 + \epsilon)$-relaxed decision procedure for target value $T$. 

Start by rounding down each $p_i$ to an integral multiple of $\epsilon^2 T$ - one of $\{\epsilon^2 T, 2\epsilon^2 T, \ldots, T\}$. Call these rounded values $p'_j$ and the modified problem $P'$ (the original problem is $P$). Observe that

$$\min_j p_j \geq \epsilon T > \epsilon^2 T \quad \text{and} \quad \max_j p_j \leq T$$

giving the number of distinct $p'_j$s as less than $\frac{T}{\epsilon^2 T} = \frac{1}{\epsilon^2}$.

The modified problem $P'$ can be solved using the dynamic programming procedure $\text{DynMakespan}_S$, since the processing times are from a finite set of values. The number of job sizes $(s)$ is now $1/\epsilon^2$ giving a running time of $O(n^2/\epsilon^2)$. The $p'_j$s also satisfy the properties

1. The rounding error $(p_i - p'_i)$ is almost $\epsilon^2 T$ for all $i$.
2. Any feasible schedule assigns no more than $1/\epsilon$ jobs on each machine (since each job is at least $\epsilon T$).

Consider $\text{DynMakespan}_S$ run on the modified problem $P'$ outlined above. If it returns No for a particular $T$, then there is no feasible solution for the “smaller” problem $P'$, and hence, there is no feasible solution for the original problem $P$ as well.

If $\text{DynMakespan}_S$ returns a Yes (and a feasible solution) for $P'$. The length of the schedule (the Makespan) is $T$, say. Then, the same schedule gives a Makespan of $(1 + \epsilon)T$ for the original problem $P$. This follows from the properties above, with each job taking almost $\epsilon^2 T$ extra time and almost $1/\epsilon$ such jobs on any machine.

Now, to handle such jobs in $P$ which have a $p_i < \epsilon T$. Add them greedily to machines whose load, as returned by $\text{DynMakespan}_S$, is $< T$. As noted before, this gives a $(1 + \epsilon)$ solution or, if such an allocation is not possible, then all the machines have load $> T$ and the Makespan of $P$ is greater than $T$ and the No returned is justified.

**Algorithm 1**  

*Input*: A set of jobs $I$ specified as the numbers $\{n_1, n_2, \ldots, n_i\}$ of jobs with corresponding processing times $\{z_1, z_2, \ldots, z_i\}$, the number of machines $m$ and a target $T$. A constant $\epsilon$

*Output*: A schedule whose Makespan is within $(1 + \epsilon)$ of the optima or a No indicating no feasible schedule with Makespan $\leq T$

1. form the set $I'' = \{\text{jobs of size } \leq \epsilon T\}$.
2. form $I' = I - I''$
3. round each job size $z_i$ to the nearest value in $\{\epsilon^2 T, 2\epsilon^2 T, \ldots, T\}$ which is less than $z_i$. call the rounded values $z'_i$.
4. run $\text{DynMakespan}_S$ on $I'$ and parameter $T$
5: if DynMakespan_S returns No then
6: return No
7: else
8: for each job $j \in I''$ do
9: if $\exists$ machine $i$ with load $\leq T$ then
10: add $j$ to $i$'s schedule
11: else
12: return No
13: end if
14: end for
15: return Yes and schedule
16: end if