Approximation Algorithms - Tools and Techniques **Spring 2002**

Lecture 6 : Jan 30, 2002

Lecturer: Kamesh Munagala [1] *Scribe: Siddartha Y R*[2]

A PTAS for the MAKESPAN Problem

The Class SNP

Definition 1 *A problem A is said to be* Strongly $\mathcal{NP}-hard$ *or* $\mathcal{SNP}$*-hard if every problem in* $\mathcal{N}P$ *can be reduced to 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 $\mathcal{SNP}$-hard problem cannot have a psuedo-polynomial time algorithm, unless $\mathcal{P} = \mathcal{NP}$. Most known $\mathcal{NP} - hard$ problems are $\mathcal{SNP}$-hard, ex. SET-COVER, TSP. The KNAPSACK problem is an example of a non $\mathcal{SNP}$-hard problem, and we saw a psuedo-poly time algorithm for it in Lecture 5. The set of problems with no psuedo-poly time algorithm is a proper subset of the the class SNP. It may be shown, under certain weak restrictions (and the assumption $\mathcal{P} \neq \mathcal{N}P$), that a problem which admits an FPTAS also admits a psuedo-poly time algorithm. This in turn implies that $\mathcal{SNP}$-hard problems do not admit an FPTAS.

Theorem 1 *The minimum* MAKESPAN *problem is* $\mathcal{SNP}$*-hard. Hence, it does not admit an FPTAS, unless* $\mathcal{P} = \mathcal{N}P$.

In this lecture, we show the following result.

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

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

[1] Dept of Comp. Sci., Stanford Univ.

[2] Dept of Comp. Sci. and Automation, Indian Institute of Science, Bangalore. `siddu@csa.iisc.ernet.in`

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 $0 < C^* < \sum_i p_i \leq np_{max}$, where p_is are the processing times, n is the number of jobs, and P_{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 $\log_2 np_{max}$ (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 algortithm) 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 Makespan(S)

Consider the problem of scheduling jobs with processing times $\in Z$ 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

$$\begin{aligned} \textstyle\sum_i v_i z_i &\leq T \\ v_i &\leq x_i \quad \forall i \end{aligned}$$

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 atmost T. Note that the solution to Makespan(S) is just $\min_{M(n_1, n_2, \ldots, n_s) \leq m} T$.

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 Makespan(S). Call this algorithm DynMakespan_S.

Relaxed Decision Procedures

Definition 4 *A ρ-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 $C \leq \rho T$.*

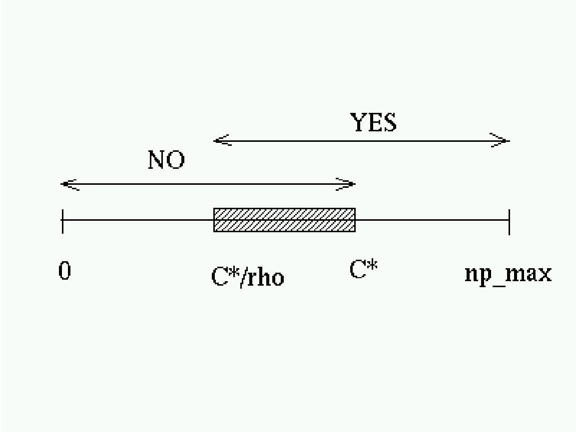

Figure 1: Output of a ρ-relaxed decision procedure for MAKESPAN(S)

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

Theorem 5 *A ρ relaxed decision procedure for* MAKESPAN*(S) yields a ρ-approximation algorithm.*

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

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

The PTAS for MAKESPAN

We solve the $(1+\epsilon)$-relaxed decision procedure for 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.

Theorem 7 *For* MAKESPAN*(S)problems with $p \geq \epsilon T$, $\forall j$, 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

$$\begin{array}{rcl} \min_j p_j & \geq & \epsilon T \;\; > \;\; \epsilon^2 T \quad \text{and} \\ \max_j p_j & \leq & T \end{array}$$

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

The modified problem P' can be solved using the dynamic programming procedure 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_is also satisfy the properties

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

Consider 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 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 atmost $\epsilon^2 T$ extra time and atmost $1/\epsilon$ 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 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 Makespan_RDP

Input : A set of jobs I specified as the numbers $\{n_1, n_2, \ldots, n_s\}$ of jobs with corresponding processing times $\{z_1, z_2, \ldots, z_s\}$, the number of machines m and a target T. A constant ϵ
Output : A schedule whose MAKESPANis 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 DynMakespan_S on I' and parameter T

```
if DynMakespan_S returns No then
  return No
else
  for each job j ∈ I'' do
    if ∃ machine i with load ≤ T then
      add j to i's schedule
    else
      return No
    end if
  end for
  return Yes and schedule
end if
```