CSC373S Lecture 5

- I was asked if I could find another time for an office hour. Would Monday at 11AM be a good hour or Tuesday at 2PM? Is T,R at 10 a bad time?

- I had an enquiry asking me where in the text is the material relating to my third lecture (on Friday, Jan 14). The answer is that it is not in the text. There will be a few things that we do that are not in the text (or previous notes).
  Lets review what we discussed in lecture 3 :

  1. charging arguments ; we started this discussion in lecture 2.
  2. chordal graphs as a generalization of interval graphs in which the greedy algorithms for (one machine) interval selection and coloring can be formulated as greedy algorithms for chordal graph MIS and (respectively) chordal graph colouring. The key idea here is the concept of a *perfect elimination order* PEO $v_1, \ldots, v_n$ of the vertices so that the "local neighbourhood" $N(v_i)$ of v_i is a clique in the graph induced by $\{v_i, \ldots, v_n\}$. Ordering intervals by non-decreasing order of finishing times is a PEO in the corresponding interval graph.
  3. a generalization of chordal graphs (which I will call inductively k-independent) that captures the JISP problem; now the local neighbourhood $N(v_i)$ of v_i (in what I called a k-PEO) has at most k independent vertices in the graph induced by $\{v_i, \ldots, v_n\}$. In fact, in many instance of such graphs, $N(v_i)$ satisfies a stronger property, namely the vertices of $N(v_i)$ are covered by k cliques. The JISP problem induces a (inductively 2-independent) graph where ordering the vertices in correspondence with non-decreasing finishing times yields an intersection graph where the local neighbourhood is covered by 2 cliques, corresponding to intervals that intersect at the finishing time and those that belong to the same job/class (choosing arbitrarily when an interval can be in both cliques). The greedy algorithm for JISP generalizes to be a k-approximation for MIS on inductively k independent graphs and hence is a 2-approximation for the JISP problem.
  4. There are benefits and costs in considering models abstracting more specific concepts. The benefits are that it is often more elegant and sometimes easier to study by (stripping away "unnecessary detail") and it can suggest further extensions. The cost is that we can lose important information. In particular, I pointed out that the best-fit EFT algorithm (which uses the geometric information of the finishing times) for m machine interval scheduling does not generalize to chordal graphs.

- Returning to the text and previous lecture notes, we will not discuss in class the Huffman prefix code (computing an optimal prefix tree) algorithm (discussed in the text - section 4.8). It can be seen as an adaptive order greedy algorithm when one thinks of the leaf nodes (ie the alphabet letters being coded) and the internal nodes of the prefix tree as the input items and the decision is to decide on the parent of a node (in the tree). This is a very practical algorithm for compression.

- One final (for the time being) application of a greedy algorithm. The problem is what is called "the makespan problem" on m identical machines or load balancing problem as referred to in the previous notes and text. It is another scheduling

problem and holds a certain historical significance. It is an NP hard problem (even for $m = 2$ machines) so that we will only attempting to find good approximation algorithms. See text section 11.1 and Approximation algorithm notes.

More specifically, in the makespan problem for m identical machines, we are given an integer parameter m and n jobs, each job $J(j)$ characterized by its run time (or load) t_j. The goal is to find a schedule $\sigma : \{1, \ldots, n\} \to \{1, \ldots m\}$ so as to minimize the *makespan* $= \max_i\{L_i\}$ where $L_i = \sum_{j:\sigma(j)=i} t_j$ is the load placed on machine i. That is, we need to schedule all jobs and they can run uninterrupted on any machine in any order taking up their time (load) and the goal is to minimize the latest finishing time on any machine.

Graham studied this problem (and an extension to machines with speeds) in two seminal papers published in 1966 and 1969 (before NP completeness was defined). Graham observed that no optimal solution seemed possible and proposed approximation algorithms. (This is considered the first paper to present a worst case approximation algorithm.) He called attention to a class of algorithms which he called "list scheduling algorithms" which is just what I have been calling (in a more general sense) fixed order greedy (or greedy-like) algorithms. Namely, these algorithms consider jobs in some order and then irrevocably place each job on the currently least loaded machine (or on some machine if we do not insist on a greedy decision as to where to schedule the job).

The first algorithm considered is the online greedy algorithm: that is, the jobs are not sorted and are considered in any order (and hence might be in the order that they arrive which is why I like to call this the online greedy algorithm) and greedily scheduled.

Theorem (Graham 1966) : The online greedy algorithm is a $(2 - 1/m)$ approximation algorithm and this bound is tight (i.e. there is a set of jobs that cause this ratio). That is, Online greedy cost $\leq (2 - 1/m)$ OPT cost.

The proof is by the same type of argument used to show that the greedy colouring (using the order $s_1 \leq s_2 \ldots \leq s_n$) is an optimal algorithm for interval colouring. Namely, we will find some parameters of the input that any solution must satisfy and then derive an upper bound on the makespan of the online greedy algorithm.

OPT (or any solution) satisfies:

1. OPT $\geq \frac{1}{m} \sum_{j:1\leq j\leq n} t_j$ since the max loaded machine must be at least as loaded as the average (over machines) load.
2. OPT $\geq \max_{j:1\leq j\leq n} t_j$ since any job load imposes a bound on the maximum load.

Now to obtain a bound on the greedy makespan, we can argue that wlg (without loss of generality), the last job $J(n)$ determines the makespan. By the greedy prop-

erty the greedy makespan is at most $\frac{1}{m}\sum_{j:j\neq n} t_j + t_n$ since the minimum machine load is at most the average machine load. Adding $\frac{1}{m}t_n$ to the sum (and then subtracting it) we have that the greedy makespan is at most $\frac{1}{m}\sum_{j:1\leq j\leq n} t_j + (1-\frac{1}{m})t_n$ which is at most OPT + $(1-\frac{1}{m})$ OPT $\leq (2-\frac{1}{m})$ OPT.

It is instructive to consider an example that forces this approximation. Consider the input consisting of $m(m-1)$ unit time jobs followed by a job with time m. Greedy will spread out the first $m(m-1)$ unit jobs evenly amongst the m machines with load $m-1$ on each machine so that the concluding makespan is $2m-1$. OPT on the other hand will put the unit jobs on $m-1$ machines and save one machine for the big job resulting in a makespan of m and hence the ratio is $\frac{2m-1}{m} = (2-\frac{1}{m})$.

Why is this instructive? Because it suggests how we might want to order the jobs (assuming that we are not in an online setting where there is no choice of order). Namely, we should take the largest jobs first. We can call this LPT-greedy makespan or just LPT (when the context is clear). I will just state without proof the following tight approximation result:

Theorem (Graham 1969): LPT is a $\frac{4}{3} - \frac{1}{3m}$ approximation algorithm.

•