CSC2420: Lecture 2

- Todays agenda:

1. Finish up PTAS for makespan
2. Introduce local search
3. Introduce IP/LP

A PTAS for all *m*

- We can think of m as being a parameter of the input instance and now we want an algorithm that is poly in *m,n* for any fixed *epsilon = 1/s*.
- We will need a combination of paradigms and techniques to achieve this PTAS.
- In particular, we will need dynamic programming (DP), greedy, scaling, and binary search.

The PTAS high level idea

- Let *T* be a candidate for an achievable makespan value. Depending on *T* and the *epsilon* required, we will scale down "large" jobs so that there are only *d = s^2* values for the job sizes. When there are only a fixed number *d* of job sizes, we use DP to test in time O(*n^{2d}*) if there is a solution that achieves makespan *T*. If there is such a solution then small jobs can be greedily scheduled without increasing the makespan too much. Use binary search to find a good *T*.

DP for fixed number of job values

- Let *z_1, …z_d* be the *d* different job sizes and let *n = n_1 + n_2 …+ n_d* be the number of inputs with *n_i* jobs having size *z_i*.
- Let *M*[*x_1,…,x_d*] = minimum number of machines needed to schedule *x_i* jobs with size *z_i* within makespan *T*. (Here we can assume *T* >= max p_i >= max z_i so that this minimum is finite.) The *n* jobs can be scheduled within makespan *T* iff *M*[*n_1, …,n_d*] is at most *m*.

Computing $M[x\_1,\ldots x\_d]$

- Let **V** = {($v\_1,\ldots,v\_d$)| $sum\_i\ v\_i\ z\_i <= T$} be the set of "configurations" that can complete on one machine within makespan T; that is, v_i jobs with size z_i on the machine. Note |***V***| <= $n\^d$
- $M(0,0,\ldots 0) = 0$
- $M(x\_1, \ldots,x\_d) = 1 + \min\_\{(v\_1,\ldots,v\_d)$ in ***V*** and $v\_i <= x\_i\}$ $M(x\_1\text{-}v\_1, \ldots,x\_d\text{-}v\_d)$
- There are at most $n\^d$ array elements and each entry uses ~ $n\^d$ time to compute (given previous entries) so that the total time is $n\^\{2d\}$.
- Must any (say DP) algorithm be exponential in d?

Large jobs and scaling (not worrying about any integrality issues)

- A job is large if $p_i >= T/s = T*epsilon$
- Scale down large jobs to have size p'_i = largest multiple of $T/(s^2)$
- $p_i - p'_i <= T/(s^2)$
- There are at most $d = s^2$ job sizes p'
- There can be at most s large jobs on any machine not exceeding target makespan T

Taking care of the small jobs and accounting for the scaling down.

- We now wish to add in the small jobs with sizes < *T/s*. We continue to try to add small jobs as long as some machine does not exceed the target makespan *T*. If this is not possible, then makespan *T* is not possible.
- If we can add in all the small jobs then to account for the scaling we note that each of the at most *s* large jobs were scaled down by at at most *T/*(*s^2*) so this only increases the makespan to (*1+1/s*)*T*.

One more “combinatorial algorithm” for this makespan problem.

- Lets consider what I called the other (than greedy and brute force search) conceptually simplest search/optimization algorithm, namely *local search*. The following local search algorithm achieves the same (*2-1/m*) approximation ratio of the online greedy alg using the same kind of analysis. In particular, the greedy solution is a local opt for this alg.

Local search algorithm for makespan

- Start with any initial solution
- While there is some job *p_i* that is now finishing last (and hence defining the makespan) and can be moved so as to finish earlier, then move this job to the least loaded machine.
- It can be shown that by moving to the least loaded machine, no job gets moved more than once and hence the algorithm has at most *n* iterations.
- The same worst case example shows that the bound is tight.

More on local search for makespan

- It doesn't matter how jobs are arranged on a machine so why not move any job (on a ``critical machine'') if that can improve things. That is, we will say that a (successful) jump move is moving any job to another machine (the result of which will either be to decrease the overall makespan or to reduce the number of machines determining the current makespan).
- The jump local search then just keeps executing successful jump moves until none possible. Finn and Horowitz (1979) prove: ``locality gap'' is $2-2/(m+1)$.

Still more on makespan local search

- The analysis here is not that different than what we used for the online greedy.
- As before we know *OPT >= (sum_i p_i)/m*
- We also know that *OPT >= max p_i* . We can further this observation by then noting that if the alg makespan = *max p_i* then the local search (or greedy) alg is optimal. So unless local search is giving an optimal answer, we must have at least 2 jobs on any machine (say *M_1*) that is determining makespan C_1. Local opt implies *C_i >= (C_1)/2*

Proof continued

- Hence *sum_j p_j = sum_i C_i*
  *= C_1 + sum_{k not 1} C_k >= C_1 *(m-1)/2 + C_1*
  = (m+1) (C_1)/2
- So *OPT >= sum_j p_j/m >= (m+1)/2m ALG* which is the desired *2-2/(m+1)* bound. For uniform machines, the locality gap is (1 + sqrt{*4m-3*})/2.
- Open: What is the best local search algorithm for this problem? One possibiity: try jump + swap neighbourhoods or cyclic neighbourhoods but these algs have the same locality gaps.

Makespan and the push neighbourhood

- This neighborhood is inspired by the Kernighan and Lin variable depth local search algorithms for graph partitioning and TSP. A push is a sequence of jumps:
- A push is initiated by a jump of a job *J_k* on a critical machine to a *M_i* on which it ``fits'' in the sense that *p_k + sum_{J_j* on *M_i* and *p_j >= p_k} p_j* is less than the current makespan. If smaller jobs on *M_i* cause load on *M_i* to equal or exceed the current makespan then in order of smallest jobs first, we keep moving small jobs to a priority queue. We then try to move jobs (in order of the largest job first) on the queue to a machine on which it fits and continue the process until either there is no machine on which it fits or the priority queue is empty.

Locality gap for push local search

- Claim: For identical machines the push local search alg has locality gap at most *4/3 – 1/(3m)* with a nearly matching lower bound of 4m/(3m+1); the LB is matched for m=2. For uniform machines, the jump locality gap is at most 2 – 2/(m+1) and the LB is arbitrarily close to 3/2.
- Push does not give constant (indep. of input values) approx for unrelated machines model which we now discuss.

Other makespan models

- In addition to the identical and uniform (aka related) machine models, there are two other well studied models for the makespan problem.
- The most general model is the unrelated machines model where now job *J_j* is a vector <*p_{j1}*, …,p{*ji*}> where *p_{ji}* is the processing time for job *j* on machine *i*.
- A special case of the unrelated machines model is the restricted machine model where now a job *J_j* is described by a pair (*p_j, S_j*) where *p_j* is the processing time for a restricted set *S_j* of machines. That is, *p_{ji}* is either *p_j* or infinity.

IP/LP rounding

- As far as I know there is no greedy, DP, local search based methods for obtaining an O(1) approximation for makespan in the unrelated or restricted machines models. Using the IP/LP rounding approach, Lenstra, Shmoys and Tardos developed a 2-approximation and a also showed it is NP hard to do better than a 1.5 approx even for the restricted machines model. This remains the state of the art.
- In fact, the IP/LP approach for makespan is a little involved so lets first consider what is perhaps the simplest example of this approach, namely weighted vertex cover.

IP/LP for vertex cover (VC)

- Input: graph $G = (V,E)$ and weight function w on vertices V. Goal: minimize the weight of a cover of the edges by the vertices; that is, a vertex cover is a subset V' of the vertices such that every edge is adjacent to at least one v in V'.
- The goal is to find such a V' so as to minimize the $sum\_\{v \in V'\} w(v)$. This is a classic NP-hard optimization problem for which there is a (non-obvious) greedy 2-approximation algorithm. (In the unweighted case, there is a simple greedy 2-approx). It is believed that a 2-epsilon approx is not possible in poly time even for unweighted case.

IP/LP for VC continued

- We want to introduce IP {0,1} variables {*x_i*} with intended meaning *x_i = 1* iff *v_i* in cover *V'*. We then need to express the objective and constraints (to enforce the intended meaning of the IP variables) using linear equations and inequalities.
- Thus we want to min *sum_i w_i x_i* subj to the linear constraints *x_i + x_j >=1* for each (*i,j*) in *E* and the integral constraint that *x_i* in {0,1}.
- Aside: common mistake in CSC 373 re IP formulations
- The LP relaxation is to allow fractional *x_i* in [0,1]. If {*x^*_i*} is an optimal LP solution then we round each *x^*_i* to *x'_i* where x'_i = 1 iff x^*_i >= ½.
- It follows that *LP-OPT <= IP-OPT = OPT <= 2* LP-OPT*

Integrality gap; returning to makespan

- The ratio (*IP-OPT/LP-OPT*) is called the integrality gap and this ratio is essentially tight for this IP/LP.
- I would call this "independent rounding" in that the rounding is done independently for each var.
- We will now consider a natural IP formulation for the makespan problem. We now want to have IP {0,1} variables x_{*ji*} = 1 iff J_j is scheduled on *M_i*.
- We then want to minimize *t* subject to:
  sum_*i* *x*_{*ji*} = 1 for each job *J_j* and
  sum_*j* *x*{ji} p_{ji} <= *T* for each machine *T*