CSC2420: Lecture 4

- Todays agenda:

1. Continue discussion of priority algorithms as a model for greedy algorithms
2. Limitations of priority algorithms
3. Extensions to the model
4. Abstracting when greedy algorithms work well

Last class we started considering a closer look at greedy and greedy like algorithms

- Our first two algorithms (the online greedy and LPT greedy) were for the makespan problem on identical machines. We want to present a model for such greedy algorithms.
- Intuitively, the algorithm sorts ("in some reasonable way") the input items (e.g. jobs to be scheduled) and then "greedily" makes an irrevocable decision (e.g. what machine to schedule the job on). The main ingrediant of this definition is what kind of sorting do we allow.

Fixed order priority algorithms

- Let ***I*** = {*I_1, I_2, …,I_n*} be the (actual) set of input items and let ***J*** be set of all possible input items. Define a *local ordering* as follows: Consider any total ordering < on ***J***. This induces a total (local) ordering on ***I***.
- Most often such a local ordering is obtained by defining any function *f:**J*** -> reals. Letting *f*(*I_j*) = *r_j* (and breaking ties say by job index), we have the local ordering induced by r_1 < r_2 < r_3. **NOTE**: the notation implies that *f*(*I_j*) does not depend on any other *I_k*.
- Clearly the LPT ordering is a local ordering.

Fixed order priority continued

- The schema of a fixed order priority algorithm:
- Sort the input set ***I*** by some local ordering and after sorting let the input items be *I_1, …,I_n*
  For *j*:= *1 ..n*
    Make an irrevocable decision *d_j* for *I_j*
  End For
- Such a schema might be best called a model for *mypoic algorithms* (myopic in the sense of not knowing the future input items at any iteration).
- We model greedy algorithms by priority algorithms in which each decision *d_j* is “greedy” in the sense of “live for today” by making a decision that optimizes the objective function without regard for any possible future items.

Greedy is an adjective : don't expect a Church-Turing like agreement.

- People use the term greedy algorithm in more general ways and in particular, the "live for today" idea for a greedy algorithm may not be at all clear or meaningful in a given situation. Hence I believe the concept of a myopic algorithm is more robust and interesting.
- We will see other greedy algorithms, some of which fit this model and some do not.

Another example of a fixed order (greedy) priority algorithm

- Consider the interval selection/scheduling problem on one or *m* machines. An input ***I*** is now a set of intervals. One reasonable representation is to represent each interval *I_j* by a pair (*s_j,f_j*) where *s_j* (resp *f_j*) is the finishing time of *I_j*. In the weighted case, *I_j* = (*s_j, f_j, w_j*) where *w_j* is the weight or value of the interval. In the one machine case, the objective is to select a maximum size (feasible) subset of non intersecting intervals. (We can allow or not allow end-points to coincide; lets say for definiteness *s_j* < *f_j* and we allow *f_j* = *s_k*.) In the weighted case we want to select a feasible subset so as to maximize the sum of weights.
- For *m* machine interval scheduling, the goal is to define a mapping sigma: {*1, …,n*} -> {*0,1, …,m*} so that intervals assigned to the same machine do not intersect.

Priority algorithms

- In any specific application, the definition of a local ordering is relative to how input items are represented. In the case of makespan there is some but not too much choice. But for interval selection/scheduling besides the geometric or interval representation, we could instead represent intervals as nodes using an adjacency list representation for the induced (interval) graph. Or we could combine these representations or just give the "degree" for each interval. Of course, the graph representation may require size ~ n^2 rather than ~ n.
- The application may also allow some freedom in the nature of the decisions. For m machine interval scheduling, it seems natural that a decision is whether or not to accept an interval and if accepting to decide which machine.

Greedy algorithm for unweighted case

- Now that there are two parameters (*s_j* and *f_j*) for each input item, the choices for how to order the inputs is less clear (even for one machine and unit weights). Three possibilities:
  - Sort so that $p_1 <= p_2 \ldots <= p_n$ where $p_j = f_j - s_j$ SPT
  - Sort so that $s_1 <= s_2 \ldots <= s_n$ EST
  - Sort so that $f_1 <= f_2 \ldots <= f_n$ EFT
- For each ordering, the relevant greedy algorithm considers each interval and accepts it iff it does not conflict with any previously accepted interval.
- Whereas SPT might seem most natural, it is not optimal but rather is a 2-approximate algorithm. EST does not achieve any constant bound and EFT is optimal.
- Two methods for proving EFT optimality
  - Charging argument
  - Inductively showing partial solutions are “promising”

Proving optimality of EFT for unweighted interval selection

- Charging argument: Let *h*: {*OPT*} -> {*EFT*} defined by *h*(*I_j*) = arg min *k* [*I_k in EFT* and *I_j* intersect]
  - *h* is a function since *I_j* must exist
  - *h* is 1-1 If not there exists one I_j with *f_j <= f_k* taken by *OPT*
- The same argument can be used to show that SPT is a 2-approximation algorithm.
- Promising partial solution argument: By induction show that *S_i*, the partial solution after *i* iterations can be extended to an optimal solution *OPT_i*.
- The algorithm (and inductive argument) can be extended to the *m* machine case, namely the algorithm is “closest fit” *EFT*. “First fit” is not optimal.

Another style for a greedy algorithm

- Consider the following greedy algorithm that adaptively chooses the next interval to consider.

S:= empty set

K:= I

While *K* not empty, select the interval *I_j* in *K* that conflicts with the fewest other intervals in *K* ;
Add *I_j* to *S* and remove all conflicting intervals in *K*

End While

This is also not an optimal algorithm but it is less obvious to precisely establish its approximation ratio (~3/2).

Weighted interval selection

- Is there an optimal (or good approximation) greedy algorithm for weighted interval selection? It is quite easy to show that there is no fixed or adaptive order priority algorithm that can achieve approximation better that *D* where *D* = max_*i*(*w_i/p_i*)/min_*i*(*w_i/p_i*). Consider the set ***J*** (of possible intervals) consisting of one long interval *I* of length and value *c^2* containing *c^2* disjoint unit intervals each of value *c*. Choosing *I* first, eliminates *c*c^2* total value. Choosing a small interval first would be equally bad if there was only one such interval.
- Note: Can modify proof so that *n* = number of intervals is known to the algorithm.
- For proportional profit (where *w_i = p_i* and hence *D =1*), LPT achieves approximation ratio *3* (and this is tight for priority algs) and hence one can always achieve *3D* by LPT.

Extensions to priority algorithms

- As suggested before, once the algorithm has access to more global information (such as say average *w_i*, min/max p_i, etc) or local information (e.g. each interval representation contains intersection information) it may be possible to do better.
- But we would like to also consider some “small ways” to extend the priority definition. (We can all debate whether or not these go beyond our intuitive meaning of greedy).

The revocable acceptance packing model

- The following type of algorithm makes sense whenever we have a packing problem or more generally whenever any subset of a feasible set is feasible.
- We obtained a bad approximation ratio for priority algorithms because they can make one (or more) poor initial choices. Bar Noy et al and Erlebach and Spieksma (ES) remedy that problem by allowing the algorithm to revoke previously accepted intervals when it considers a new interval that is sufficiently (as parameterized by some factor $\alpha < 1$) more valuable.

"Greedy_a"

- Again sort intervals so that *f_1 <= f_2 …<=f_n*

S:= empty set

For *j : 1 ..n*

  Let *C_j* = intervals in *S* that conflict with *I_j*

  If *w(C_j) <= a w(I_j)*

    then *S:= S* – {*I in C_j*} + {*I_j*}

  End If

End For

Claim: *Greedy_a* achieves (tight) appoximation *1/a*(*1-a*) which is then maximized by choosing *a* = ½ resulting in a 4-approximation.

Proof sketch of *Greedy_α* approximation

- Let S^* be final set accepted and T be the set of intervals that are in S at any time. The proof consists of two observations:
  - $W(T) <= w(S^*)/(1-\alpha)$
  - $W(OPT) <= w(T) / \alpha$
- The first observation follows from the algorithm being "cautiously greedy" and not revoking intervals unless something much better occurs.
- The second observation is a charging argument.
- This is a myopic algorithm but would we call it greedy?

Other orderings possible but still ..

- Horn shows that we can order intervals so that *w_1/p_1 >= …>= w_n/p_n* and a similar proof will also achieve a constant approximation. (For some special cases this ordering obtains a better approximation.)
- But Horn also shows that no priority algorithm with revocable acceptances can achieve optimality obtaining lower bound of ~1.17

Priority stack (packing) algorithms

- We consider another more general way (due to Bar Noy et al and Berman and das Gupta) to remedy bad initial decisions.
- We extend (one pass) priority algorithms by a restricted class of two pass algorithms. Namely, in the first pass, we proceed like a priority algorithm except irrevocable acceptances are now replaced by “push interval onto the stack”; the second stage then pops the stack to insure feasibility. That is, while popping the stack, we place an interval in the solution S if it does not conflict with any intervals already placed in S.
- Claim: If we again sort by finishing times, there is an optimal stack algorithm. Namely, we will maintain *current interval weights* and only push an interval if it has positive current weight. In doing so, we subtract this current weight from all (later) conflicting intervals.

Proof sketch for priority stack optimality

- Let *S** be final set accepted, *T* be the set of intervals that are in the stack at the end of the push stage and let *T_j* be the intervals on the stack just before *I*_j in *S** is pushed . The proof consists of two observations:
  - *w(S*) >= w(T)*
  - *w(OPT) <= w(T)*
- Note: This stack algorithm is an example of a "local ratio" packing algorithm (not local search) and local ratio algorithms are informally equivalent to primal dual algorithm as discussed in Bar Yehuda and Rawitz. In particular, the stack algorithm for weighted interval selection can be viewed as a primal dual algorithm with reverse delete (as discussed in Williamson).

Optimal greedy interval colouring

- Whereas the greedy algorithm utilizing the ordering *s_1 <= s_2 …<= s_n* has an arbitrarily bad way approximation ratio, the problem of interval colouring is optimally solved by greedily using this ordering; that is, if the colours are 1,2, … the algorithm colours *I_j* using (say) the smallest non conflicting colour (introducing a new colour only whenever necessary).

Proof of greedy colouring optimality

- The proof follows the style used for the greedy makespan algorithm. Namely, when viewed as a graph colouring problem (in this case we have an interval graph),
  - it is clear that the chromatic number (i.e. the minimum number of colours needed) is at least as large as the graphs clique number (i.e. the size of a maximum clique).
  - The greedy colouring (with the EST ordering) will only introduce colour *k* if there is a clique of size *k*
- As a byproduct of this algorithm/proof, we obtain the result that interval graphs are “perfect graphs”.