CSC2420: Algorithm Design, Analysis and Theory
Fall 2023
An introductory (i.e. foundational) level graduate course.

Allan Borodin

October 3, 2023

Week 4

Announcements:

- Would anyone like to have Victor do a TA office hour answering questions about Assignment 1 or other questions?
- in question 3 of the Assignment, when I said $k-1$ mapping, I meant k to 1 mapping.

Todays agenda
We will discuss the following topics:

- The charging argument proofs for the unweighted and proportionally weighted interval selection problems. I put those proofs in the slides for Week 3 but also repeating the slides today since we didn't do the proofs pertaining to priority algorithms for interval selection.
- Greedy algorithms for the set packing problem
- Maximum weighted independent set in a matroid

Charging arguments for the unweighted and proportionally weighted interval selection problem.

Let first consider the unweighted interval selection problem where each input item is an interval $I_j = [s_i, f_i)$ with s_i (resp. f_i) the starting time (finishing time) of the interval. The greedy algorithm sorts the intervals so that $f_1 \leq f_2 \ldots \leq f_n$. We will construct a 1-1 charging function $h : OPT \rightarrow Greedy$ (which implies $|OPT| \leq |Greedy|$) as follows:

We charge any interval in $OPT \cap Greedy$ to itself. So now we need to charge an interval I_j in $OPT \setminus Greedy$ to an interval selected by the greedy algorithm. The function h will charge I_j to the leftmost interval I_i with which it conflicts. Since $I_j \notin Greedy$, it must be that I_i exists and $f_i \leq f_j$ or else Greedy would have taken I_j. This shows that h is well defined. To show that h is 1-1, we need to show that there cannot be two OPT intervals being charged to I_i. But any I_k with $k \geq i$ must intersect I_i at f_i which means that I_j and I_k would intersect.

The job interval scheduling problem and the greedy algorithm

In the NP-hard job interval scheduling problem, every interval belongs to exactly one job class and now a feasible solution means that accepted intervals do not intersect and at most one interval from any job class is accepted.

We again sort the intervals so $f_1 \leq f_2 \ldots \leq f_n$. We use essentially the same charging argumnent except now the charging function h is a 2-1 function which implies that $|OPT| \leq 2|Greedy|$. That is an interval $I_j \in OPT \setminus Greedy$ either intersects an interval I_i or I_j and I_i are in the same job class.

A charging argument for the proportionally weighted interval selection problem

Here we assumne that the weight (i.e., the profit) of a scheduled interval is equal to its length $f_i - s_i$. The claim is that LPT (Longest Processing Time first) provides a $\frac{1}{3}$ approximation. Moreover, this is the optimal approximation for any priority algorithm; more specifically $\frac{1}{3} + \epsilon$ for any $\epsilon > 0$.

We use a charging argument to establish the positive result. Namely, we provide a charging function $f; OPT \to LPT$ such that f charges at most 3 times the weight of intervals in OPT to an interval in LPT.

Once again if $I \in OPT \cap LPT$, then charge I to itself. Otherwise consider an interval $I \in OPT \setminus LPT$. Charge I to the interval I' in LPT that intersects with I having the earliest finishing time. (Here we could charge I to I' with the earliest starting time or really charge in any way that provides a unique I'.). We know that such an I' must exist or else LPT (being greedy) would have taken the interval I.

Completing the charging argument for proprtionally weighted interval scheduling

I can intersect I' by overlapping at an endpoint s_i and/or f_i of I' or it can be that I is contained in I'. We can deal with each of these cases.

- I subsumes I'. This can't happen since the the length (and therefore profit) of I would be more than I' and hence LPT would have taken it.
- I is subsumed by I'. The sum of the lengths of all such intervals I is at most the length of I'.
- I' intersects at the start time s_i (or finishing time f_i) of I'. There can be at most one interval in OPT intesecting at s_i (resp. f_i). By the LPT ordering, $|I| \leq |I'|$.

The negative result for proportional weighted interval selection fn priority algorithms

For the negative result consider input instances where we have long jobs (as depicted in Figure 1) and enough short jobs to fill up any long job. Each small job has negligible size compared to the interval that subsumes it. Any priority algorithm must take the first interval I it decides to process or else the adversary stops the input sequence and OPT takes I.

If it is a small job, the adversary just gives the long the long job subsuming it.

If I is a large job $J_k \neq J_1$, the adversary gives all the small jobs within I and the two adjacent intervals.

If I is a large job $J_k = J_1$, the adversary gives all the small jobs within I and the one adjacent interval.

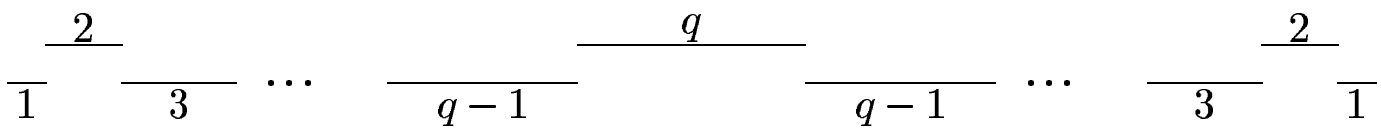

Figure 1: The "long jobs" from the worst case sequence for any priority algorithm for $m = 1$.

Interval scheduling on more than one machine

I say scheduling (and not selection) when considering more than one machine since the algorithm must not only decide which intervals to take but also indicate on which machine to place the interval. Of course, the intervals on each machine must not intersect.

Note that the job interval scheduling problem subsumes the problem of interval scheduling on many machines if we know that ltest completion time for all intervals.

The following table shows some surprises when there is more than one machine.

Proportional profit	FIXED PRIORITY	ADAPTIVE PRIORITY
GREEDY	all m $\rho = 3$	m even $\rho \leq 2$ $m = 2$ $1.56 \leq \rho$
(not necessarily greedy)	$m = 2$ $2 \leq \rho$	$m = 1$ $\rho = 3$

Greedy algorithms for the set packing problem

One of the new areas in theoretical computer science is algorithmic game theory and mechanism design and, in particular, auctions including what are known as *combinatorial auctions*. The underlying combinatorial problem in such auctions is the set packing problem.

The set packing problem

We are given n subsets $S_1, \ldots, S_n$ from a universe U of size m. In the weighted case, each subset S_i has a weight w_i. The goal is to choose a disjoint subcollection $\mathcal{S}$ of the subsets so as to maximize $\sum_{S_i \in \mathcal{S}} w_i$. In the s-set packing problem we have $|S_i| \leq s$ for all i.

- This is a well studied problem and by reduction from the max clique problem, there is an $m^{\frac{1}{2}-\epsilon}$ hardness of approximation assuming $NP \neq ZPP$. For s-set packing with constant $s \geq 3$, there is an $\Omega(s/\log s)$ hardness of approximation assuming $P \neq NP$.
- We will consider two "natural" greedy algorithms for the s-set packing problem and a non obvious greedy algorithm for the set packing problem. These greedy algorithms are all fixed order priority.

The first natural greedy algorithm for set packing

Greedy-by-weight ($Greedy_{wt}$)

Sort the sets so that $w_1 \geq w_2 \ldots \geq w_n$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

- In the unweighted case (i.e. $\forall i, w_i = 1$), this is an online algorithm.
- In the weighted (and hence also unweighted) case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- The approximation bound can be shown by a charging argument where the weight of every set in an optimal solution is charged to the first set in the greedy solution with which it intersects.

The second natural greedy algorithm for set packing

Greedy-by-weight-per-size

Sort the sets so that $w_1/|S_1| \geq w_2/|S_2| \ldots \geq w_n/|S_n|$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

- In the weighted case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- For both greedy algorithms, the approximation ratio is tight; that is, there are examples where this is essentially the approximation. In particular, these algorithms only provide an m-approximation where $m = |U|$.
- We usually assume $n >> m$ and note that by just selecting the set of largest weight, we obtain an n-approximation. So the goal is to do better than $\min\{m, n\}$.

Improving the approximation for set packing

- In the unweighted case, greedy-by-weight-per-size (and greedy-by-weight-per squareroot-size) can be restated as sorting so that $|S_1| \leq |S_2| \ldots \leq |S_n|$ and it can be shown to provide an $\sqrt{m}$-approximation for set packing.
- On the other hand, greedy-by-weight-per-size does not improve the m-approximation for weighted set packing.

Greedy-by-weight-per-squareroot-size; Gonen and Lehmann EC00

Sort the sets so that $w_1/\sqrt{|S_1|} \geq w_2/\sqrt{|S_2|} \ldots \geq w_n/\sqrt{|S_n|}$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

Theorem: Greedy-by-weight-per-squareroot-size provides a $2\sqrt{m}$-approximation for the set packing problem. And as noted earlier, this is asymptotically the best possible approximation assuming $NP \neq ZPP$.

Another way to obtain an $O(\sqrt{m})$ approximation

There is another way to obtain the same aysmptototic improvement for the weighted set packing problem. Namely, we can use the idea of partial enumeration greedy; that is somehow combining some kind of brute force (or naive) approach with a greedy algorithm.

Partial Enumeration with Greedy-by-weight ($PGreedy_k$)

Let Max_k be the best solution possible when restricting solutions to those containing at most k sets. Let G be the solution obtained by $Greedy_{wt}$ applied to sets of cardinality at most $\sqrt{m/k}$. Set $PGreedy_k$ to be the best of Max_k and G.

- Theorem: $PGreedy_k$ achieves a $2\sqrt{m/k}$-approximation for the weighted set packing problem (on a universe of size m)
- In particular, for $k = 1$, we obtain a $2\sqrt{m}$ approximation and this can be improved by an arbitrary constant factor $\sqrt{k}$ at the cost of the brute force search for the best solution of cardinality k; that is, at the cost of say n^k.

Combinatorial auctions

For those who are interested, a combinatorial auction is one where n agents (buyers) want one of possibly many subsets of items $S_1, S_2, \ldots, S_k$. Each agent has has a private value $w(S_i)$ for any desired set. An auctioneer wants to allocate at most one subset to each agent so as to maximize the total value of disjoint sets that are allocated. This is called the "social welfare" objective.

We assume "free disposal" (as in the display ads problem) so that $w(S) \leq w(T)$ if $S \subset T$.

If every agent was truthful about their valuations, this would be the set packing problem. But agents are self interested and may not truthfully report the values for the sets they desire if they think it will be helpful. That is, they could over state or understate their valuations.

In order to incentivize agents to be truthful, the auctioneer decides on prices for each agent.

Combinatorial auctions continued

If we could compute an optimal solution we would have a way to speficy a price for the subset to be allocated to each agent so that the agent would be truthful. But set packing is an NP hard approximation problem as we have indicated. This result (called VCG auctions) does not hold for approximation algorithms.

Perhaps the most prominent question in CAs, is how how much truthfulness can hurt the approximation ratio for a given CA if we insist upon efficient polynomial allocations or insist upon conceptually simple allocations and pricing.

Although we need to be more precise, truthful auctions using priority algorithm allocations for say an s-CA will result in asymptotically the worst possible approxiamtion (i.e., $\Omega(\min\{n.m\}$.

Max weighted independent set in a matroid.

We conclude our current discussion of priority algorithms by mentioning a classic result about greedy algorithms and matroids.

Let U be a set of elements and $\mathcal{I}$ be a collection of subsets of U. $(U, \mathcal{I})$ is a matroid if the following hold:

- (Hereditary property) If $I \in \mathcal{I}$ and $I' \subset I$, then $I' \in \mathcal{I}$.
- (Exchange property) If $I', I \in \mathcal{I}$ and $|I'| < |I|$, then $\exists u \in I \setminus I'$ such that $I' \cup \{u\} \in \mathcal{I}$.

An *hereditary set system* $(U, \mathcal{I})$ is any set system satisfying the hereditary property so that a matroid is an hereditary set system that also satisfies the exchange property.
The sets $I \in \mathcal{I}$ are referred to as the *independent sets*. We note that there are alternative equivalent definitions. In particular, an alternative to the exchange property is that every maximal independent set is called a *basis* and has the same size. This maximum size is call the *rank* of the matroid.

Matroids and the greedy algorithm

Any acyclic subset of edges in a graph is an independent set in a matroid. If the graph is connected then a maximal acyclic subset is a spanning tree. This is called a graphic matroid.

An independent set of vectors in a vector space is a matroid. ‘ Whitney

[1935] defined this elegant abstraction that applies to many other systems. Generalizations to intersections of matroids and more general indepedence systems are also known.

The minimum spanning tree (MST) problem can be seen as a maximizaton problem (negate the signs of the edge weights or subtract each edge weight from the maxium edge weight).

Kruskal's MST algorithm can be seen as a the natural greedy algorithm for computing a maximum weight independent set in a graphic matroid.

The Rado-Edmonds theorem

The proof of the optimality of Kruskal's algorithm can be carried over to show how the natural greedy algorithm obtains an optimal solution for computing an optimal independent set **in any matroid**.
What is this natural greedy algorithm for computing a max weight independent set in a matrod?

Remarkably, there is what can be seen as a converse to this fact.

Consider any hereditary set system. If the natural greedy algorithm produces an optimal solution for every linear function of elements in an independent set, then the set system is matroid.

Does this define greedy algorithms?

The Rado-Edmonds theorem

The proof of the optimality of Kruskal's algorithm can be carried over to show how the natural greedy algorithm obtains an optimal solution for computing an optimal independent set **in any matroid**.
What is this natural greedy algorithm for computing a max weight independent set in a matrod?

Remarkably, there is what can be seen as a converse to this fact.

Consider any hereditary set system. If the natural greedy algorithm produces an optimal solution for every linear function of elements in an independent set, then the set system is matroid.

Does this define greedy algorithms?

Is the optimality of greedy algorithms limited to independent sets in matroids?
A non interecting set of intervals can be viewed as a hereditary set system but not a matroid.

Local search: the "other" conceptually simplest paradigm

Along with greedy and greedy-like algorithms, *local search* is (for me) one of the two conceptually simplest search/optimization paradigms. Like greedy algorithms, there are many variations of this paradigm.

The vanilla local search paradigm

"Initialize" S
While there is a "better" solution S'
in the "local neighbourhood" $Nbhd(S)$
$S := S'$
End While

If and when the algorithm terminates, the algorithm has computed a *local optimum*.

Local search as a well defined algorithm

To make local search a precise algorithmic model, we have to say:

1. How are we allowed to choose an initial solution?
2. What constitutes a reasonable definition of a local neighbourhood?
3. What do we mean by "better"?

Answering these questions (especially as to defining a local neighbourhood) will often be quite problem specific.

Towards a more precise definition for local search

- We clearly want the initial solution to be efficiently computed and to be precise we can (for example) say that the initial solution is a random solution, or a greedy solution or adversarially chosen.
  Of course, in practice we can use any efficiently computed solution.
- We want the local neighbourhood $Nbhd(S)$ to be such that we can efficiently search for a "better" solution (if one exists).
  1. In many problems, a solution S is a subset of the input items or equivalently a $\{0,1\}$ vector, and in this case we often define the $Nbhd(S) = \{S' | d_H(S, S') \leq k\}$ for some "small" k where $d_H(S, S')$ is the Hamming distance.
  2. More generally whenever a solution is a vector over a small domain D, we can use Hamming distance to define a local neighbourhood. Hamming distance k implies that $Nbhd(S)$ can be searched in at most time $|D|^k$.
  3. We can view Ford Fulkerson flow algorithms as local search algorithms where the (possibly exponential size but efficiently search-able) neighbourhood of a flow solution S are flows obtained by adding an augmenting path flow.

What does "better" solution mean? Oblivious and non-oblivious local search

- For a search problem, we would generally have a non-feasible initial solution and "better" can then mean "closer" to being feasible.
- For an optimization problem it usually means being an improved solution which respect to the given objective. For reasons I cannot understand, this has been termed *oblivious* local search. I think it should be called greedy local search.
- For some applications, it turns out that rather than searching to improve the given objective function, we search for a solution in the local neighbourhood that improves a related potential function and this has been termed non-oblivious local search.
- In searching for an improved solution, we may want an arbitrary improved solution, a random improved solution, or the best improved solution in the local neighbourhood.
- For efficiency we sometimes insist that there is a "sufficiently better" improvement rather than just better.

Exact Max-k-Sat

- **Given:** An exact k-CNF formula

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$

where $C_i = (\ell_i^1 \vee \ell_i^2 \ldots \vee \ell_i^k)$ and $\ell_i^j \in \{x_k, \bar{x}_k \,|\, 1 \leq k \leq n\}$.

- In the weighted version, each C_i has a weight w_i.

- **Goal:** Find a truth assignment τ so as to maximize

$$W(\tau) = w(F \,|\, \tau),$$

the weighted sum of satisfied clauses w.r.t the truth assignment τ.

- It is NP hard to achieve an approximation better than $\frac{7}{8}$ for exact Max-3-Sat and hence that hard for the non exact version of Max-k-Sat for $k \geq 3$.

- Max-2-Sat can be approximated to within a factor $\approx .87856$.

The natural oblivious local search

- A natural oblivious local search algorithm uses a Hamming distance d neighbourhood:
  $N_d(\tau) = \{\tau' : \tau \text{ and } \tau' \text{ differ on at most } d \text{ variables } \}$

Oblivious local search for Exact Max-k-Sat

Choose any initial truth assignment τ
WHILE there exists $\hat{\tau} \in N_d(\tau)$ such that $W(\hat{\tau}) > W(\tau)$
    $\tau := \hat{\tau}$
END WHILE

How good is this oblivious local search algorithm?

- Note: Following the standard convention for Max-Sat, I am using approximation ratios < 1.
- It can be shown that for $d = 1$, the approximation ratio for Exact-Max-2-Sat is $\frac{2}{3}$.
- In fact, for every exact 2-Sat formula, the algorithm finds an assignment τ such that $W(\tau) \geq \frac{2}{3} \sum_{i=1}^{m} w_i$, the weight of all clauses, and we say that the "totality ratio" is at least $\frac{2}{3}$.
- More generally for Exact Max-k-Sat the ratio is $\frac{k}{k+1}$.
  This ratio is essentially a tight ratio for any $d = o(n)$.
- This is in contrast to an online greedy algorithm derived from a naive randomized algorithm that achieves totality ratio $(2^k - 1)/2^k$.
- "In practice", the local search algorithm often performs better than the naive greedy and one could always start with the greedy algorithm and then apply local search.

Analysis of the oblivious local search for Exact Max-2-Sat

- Let τ be a local optimum and let
  - S_0 be those clauses that are not satisfied by τ
  - S_1 be those clauses that are satisfied by exactly one literal by τ
  - S_2 be those clauses that are satisfied by two literals by τ

- Let $W(S_i)$ be the corresponding weight.

Analysis of obvlivious Exact-Max-2-Sat local search continued

- We will say that a clause involves a variable x_j if either x_j or $\bar{x}_j$ occurs in the clause. Then for each j, let
- A_j be those clauses in S_0 involving the variable x_j.
- B_j be those clauses C in S_1 involving the variable x_j such that it is the literal x_j or $\bar{x}_j$ that is satisfied in C by τ.
- C_j be those clauses in S_2 involving the variable x_j.
- Let $W(A_j), W(B_j), W(C_j)$ be the corresponding weights.

Analysis of the oblivious local search (continued)

- Summing over all variables x_j, we get
- $2W(S_0) = \sum_j W(A_j)$ noting that each clause in S_0 gets counted twice.
- $W(S_1) = \sum_j W(B_j)$
- Given that τ is a local optimum, for every j, we have

$$W(A_j) \leq W(B_j)$$

  or else flipping the truth value of x_j would improve the weight of the clauses being satisfied.

- Hence (by summing over all j),

$$2W_0 \leq W_1.$$

Finishing the analysis

- It follows then that the ratio of clause weights not satisfied to the sum of all clause weights is

$$\frac{W(S_0)}{W(S_0)+W(S_1)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)}$$

- It is not easy to verify but there are examples showing that this $\frac{2}{3}$ bound is essentially tight for any N_d neighbourhood for $d = o(n)$.

- It is also claimed that the bound is at best $\frac{4}{5}$ whenever $d < n/2$. For $d = n/2$, the algorithm would be optimal.

- In the weighted case, we have to worry about the number of iterations. And here we can speed up the termination by insisting that any improvement has to be sufficiently better.

Using the proof to improve the algorithm

Aside: Using adversarial examples and viewing algorithms as a *game* against an adversary is an idea that is now very active in "adversarial learning".

- We can learn something from this proof to improve the performance.
- Note that we are not using anything about $W(S_2)$.
- If we could guarantee that $W(S_0)$ was at most $W(S_2)$ then the ratio of clause weights not satisfied to all clause weights would be $\frac{1}{4}$.
- Claim: We can do this by enlarging the neighbourhood to include $\tau' =$ the complement of τ.

The non-oblivious local search

- We consider the idea that satisfied clauses in S_2 are more valuable than satisfied clauses in S_1 (because they are able to withstand any single variable change).

- The idea then is to weight S_2 clauses more heavily.

- Specifically, in each iteration we attempt to find a $\tau' \in N_1(\tau)$ that improves the potential function

$$\frac{3}{2}W(S_1) + 2W(S_2)$$

instead of the oblivious $W(S_1) + W(S_2)$.

- More generally, for all k, there is a setting of scaling coefficients $c_1, \ldots, c_k$, such that the non-oblivious local search using the potential function $c_1 W(S_1) + c_2 W(S_2 + \ldots + c_k W(S_k)$ results in approximation ratio $\frac{2^k-1}{2^k}$ for exact Max-k-Sat.