CSC2421: Online and Other Myopic Algorithms
Fall 2025

Allan Borodin

October 1, 2025

Week 5

Annoucements

- I have posted the first assignment.. Note that there are many exercises and open problems at the end of most chapters. Please let me know if any of the exercises in the text are not clear. This problem set is due Octoberr 22 and will count for 35% of the final grade. .
  We will have another problem set due November 28 and will also count for 35% of the final grade.
- The third part of the grading scheme is a project report and short presentation on a topic within the scope of this course. This can be done individually or in pairs. I recommend doing the project with someone. The project report will be due November 21 and the presentations will begin on November 21
  I want to approve each project to insure that there isn't too much overlap between projects. You are not restricted to topics in the text but the project has to be directly related to the course.

Todays agenda

- Some followup comments regarding David Zhang's presentation.
- Extensions of online and priority algorithms. Parallel and Multipass algorithms.
- Priority algorithms for Set Packing. (Section 18.3.5 of text)
- MaxSat (chapter 11): Th naive algorithm and its de-randomization to become Johnson's algorithm; an improved randomized algorithm and its de-randomization into a two-pass algorithm.
- Experimental results for Max-Sat.

NOTE: I am departing from the sequence of chapters in the text to gain a better sense of what we mean by "online and other myopoic algorithms". And the results today are all a little surprising.

Some followup comments relating to David Zhang's presentation

The simulation of 1-bit barely random online algorithms by det algorithms in the ROM model raises many other general questions regarding the relation betwewen different myopic models.

- The $\frac{1}{2}$ competitive barely randm algorithm for the general knapsack can be thought of as a modification of the barely random priority algorithm which randomly chooses between greedily sorting by value and and greedily sorting by value-density.
  When can we simulate priority algorithms by online algorithms with revoking? Ben Cookson has some ideas in this regard.
  Note: There is an optimal fixed priority algorithmm for unweighted interval selection (order by finishing times) whereas in the online (but not real time) model, it is not possible to obtain a constant ratio by an online algorithm with revoking.
- The converse of the previous quesion is "when can we simulate online with revoking by priority without revoking?"

Folowup comments continued

- When can we simulate priority algorithms by ROM algorithms with revoking?
  For the unweighted interval selection problem, we cannot obtain optimality by a ROM algorithm with revoking but we can get a 2.5 competitive ratio and 12/11 is so far the only negative result.
- With regard to the converse, Nicholas Pena has shown that we cannot aymptotically beat the 1/2 ratio for unweighted bipartite matching by a determinitsic priority algorithm but we know we can achieve a $1 - \frac{1}{e} \approx .623$ ratio by a deterministic algorithm in the ROM model.
- In general, what does it mean an algorithm to "simulate" another algorithm?
  More specifically, what does it mean to "de-randomize" an algorithm.
- How many random bits do we need for an online algorithm proecessing a sequence of n input items?

Extensions of online and priority algorithms

In discussing more general online frameworks, we already implicitly suggested some extensions of the basic priority model (that is, the basic model where we have one-pass and irrevocable decisions). The following online or priority algorithm extensions can be made precise:

- Decisions can be *revocable* to some limited extent or at some cost. For example, we know that in the basic priority model we cannot achieve a constant approximation for weighted interval scheduling. However, if we are allowed to revoke accepted intervals (while always maintaining a feasible solution), then we can achieve a 4-approximation. (but provably not optimality). The 4-approximation is a result due to Erlebach and Spieksma
- While the knapsack problem cannot be approximated by a deterministic priority algorithm to within any constant, we can achieve a 2-approximation by taking the maximum of 2 greedy algorithms. More generally we can consider some "small" number k of priority (or online) algorithms and take the best result amongst these k algorithms.

Extensions of the priority order model continued

- Closely related to the "best of k online (or priority)" model is the concept of online (preiority) algoitthms with "advice". There are two trusted advice models, a model where one measures the maximum number of advice bits per input item, and a model where we are given some number ℓ of advice bits at the start of the computation. The latter model is what I will mean by "online with advice." Online with ℓ advice bits is equivalent to the max of $k = 2^{\ell}$ algorithms.

Extensions of the priority order model continued

- Closely related to the "best of k online (or priority)" model is the concept of online (preiority) algoitthms with "advice". There are two trusted advice models, a model where one measures the maximum number of advice bits per input item, and a model where we are given some number ℓ of advice bits at the start of the computation. The latter model is what I will mean by "online with advice." Online with ℓ advice bits is equivalent to the max of $k = 2^\ell$ algorithms.

  NOTE: This model is a very permissive in that the advice bits can be a function of the entire input. Of course, in practice we want these advice bits to be "easily determined or known a-priori" (e.g., the number of input items, or the ratio of the largest to smallest value) but in keeping with the information theoretic perspective of online and priority algorithms, one doesn't impose any such restriction.
- There are more general parallel priority based models than "best of k" algorithms. Namely, parallel algorithms could be spawning or aborting threads (as in the pBT and oBT models that we discuss later).

Multipass algorithms

- Another model that provides improved results is to allow multiple passes (over the input items) rather than just one pass.
- This is not a well studied model but there are two relatively new noteworthy results that we could discuss:
  1. There is deterministic 3/4 approximation for weighted Max-Sat that is achieved by two "online passes" (i.e., the input sequence is determined by an adversary) over the input sequence whereas there is evidence that no one pass deterministic online or priority algorithm can acheive this ratio.
  2. There is a $\frac{3}{5}$ approximation for bipartite matching that is achieved by two online passes whereas no deterministic online or priority algorithm can do asymptotically better than a $\frac{1}{2}$ approximation.
- It is not clear how best to formalize these multi-pass algorithms. Why?

Multipass algorithms

- Another model that provides improved results is to allow multiple passes (over the input items) rather than just one pass.
- This is not a well studied model but there are two relatively new noteworthy results that we could discuss:
  1. There is deterministic 3/4 approximation for weighted Max-Sat that is achieved by two "online passes" (i.e., the input sequence is determined by an adversary) over the input sequence whereas there is evidence that no one pass deterministic online or priority algorithm can acheive this ratio.
  2. There is a $\frac{3}{5}$ approximation for bipartite matching that is achieved by two online passes whereas no deterministic online or priority algorithm can do asymptotically better than a $\frac{1}{2}$ approximation.
- It is not clear how best to formalize these multi-pass algorithms. Why? What information should we be allowed to convey between passes?

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; i.e. value density

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 often 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 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

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.

(Weighted) Max-Sat

A Boolean formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses, i.e., $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause $C_i = \bigvee_{j=1}^{i_k} \ell_{i,j}$ is a disjunction of literals where a literal is a propostional variable or its complmenet. A CNF formula ϕ is called a k-CNF formula (respectively, an exact k-CNF formula) if every clause in ϕ is of length at most (respectively, exactly) k. In a *weighted* CNF formula there is a non-negative weight w_i associated with every clause C_i. We assume that a clause contains only one literal involving a given variable x.

Given a CNF formula F, the (weighted) Max-Sat problem is to select truth values for the variables in F so as to maximize the (weight) of clauses that can be simultaneously satisfied.

Propostional variables as the input items and how they are represented

Depending on what information about clauses is available in an input item, we distinguish 4 different input models numbered 0 to 3.

- (Input model 0): For each x_i, the information is the names and weights of the clauses in which x_i occurs positively and the names and weights of clauses in which x_i appears negatively.
- (Input model 1): Input model 0 plus the lengths of those clauses.
- (Input model 2): Input model 0 plus the names of the other variables occurring in each of those clauses but not their signs.
- (Input model 3): A complete description of each of the clauses in which x_i occurs.

Clearly, input model 3 is the most general input representation and input model 0 is effectively a minimal representation. In the weighted version of the problem, we also learn the weight of each clause where x_i appears.

The naive randomized online Max-Sat algorithm

The naive randomized algorithm simply chooses the truth value for each $v(x_i) \in \{True, False\}$ randoml and independently with probability $\frac{1}{2}$. The algorithm uses input model 0.

For *exact* Max-k-Sat, the *totality ratio* of the naive algorithm is $1 - \frac{1}{2^k}$. The totality ratio compares the weight of satisfied clauses to the sum of all clause weights. Clearly the competitive ratio can only be better.

The naive algorithm, can be de-randomized by the method of conditional expectation. Since the expected behaviour is "good", then for each variable, we can determine the effect (i.e., the change in the expected value) of setting the variable to be True or False and choose the setting (i.e the branch of the randomzation tree) that has the better expectation. The result obtains at least the ratio of the naive algorithm but could be better.

Note that because ot unit clauses (with one literal), the expectation will only be $\frac{1}{2}$. But of course, unit clauses should somehow be "easy" to handle.

The de-randomized nave algorithm is Johnson's algorithm

Johnson gave a determnistic algorithm for Max-Sat in 1974 that maintains, a modified weight $w'_j = w_j/2^{|C_j|}$ where $|C_j|$ is the current number of literals in C_j. That is, when we set a variable, a clause is either satsified or one literal is elimniated. This insures that clauses become less valuable (in terms of expected value) whenever a literal is eliminated. Joshnson's algorithm only needs input model 1.

In 1992, Yannakakis showed that the de-randomization of the naive algorithm is equivalent to Johnson's algorithm and showed that the competitive ratio of Johnson's algorithm cannot be better than $\frac{2}{3}$.

In 1999, Chen et al showed that Johnson's algorithm achieves the competitive ratio $\frac{2}{3}$.

A Randomized Max-Sat Algirithm with Competitive Ratio $\frac{3}{4}$

The underlying idea for improving the $\frac{2}{3}$ ratio of Johnson's deterministic algorithm is that in setting the variables, we want to balance the weight of clauses satisfied with that of the weight of clauses that can no longer be satisfied. Here we assume that W, the sum of the weights of all clauses is initially known. (I wonder if this assumption is necessary.)

The improved algorithm uses input model 2 (see Section 11.3). The first $\frac{3}{4}$ competitive algorithm was given by Polyczek and Schnitger in 2011 and then alternative algorithms due to Buchbinder et al and van Zuylen appearded culminating in the version (we present in the text) in a 2017 paper by Polyczek et al.

The $\frac{3}{4}$ ratio is best possible for any online algorithm in input model 2. This leaves open the optimal deterministic and randomized ratios for input model 3. Yung [unpublished but verified] shows that $\frac{5}{6}$ is a competitive limitation for any online Max-Sat algritihm with input model 3. The best offline approximation (via SDP) is not much better than the $\frac{3}{4}$ ratio.

Very brief sketch of the Max-Sat randomized algorithm in Polyczek et al

Let $\tau_{\leq i}$ be the assignment to the first i variables and let SAT_i be the weight of satisfied clauses with respect to the partial assignment $\tau_{\leq i}$. Let $UNSAT_i$ be the weight of the clauses that can no longer be satisfied given the assignment $\tau_{\leq i}$; that is, the clauses that are unsatisfied by $\tau_{\leq i}$ and containing only variables in $\{x_1, \ldots, x_i\}$.

We define $B_i = \frac{1}{2}(SAT_i + W - UNSAT_i)$ where W is the total weight of all clauses. Note that $SAT_0 = UNSAT_0 = 0$ so that $B_0 = \frac{1}{2}W$; note also that $SAT_n = W - UNSAT_n =$ the weight of the clauses satisfied by the algorithm upon termination which implies $B_n = SAT_n$. Also, we have $SAT_i \leq B_i \leq W - UNSAT_i$ for all i.

Continued sketch of the Max-Sat algorithm

As in the de-randomization analysis of the naive randomized algorithm, an online algorithm can calculate the changes in SAT_i and $UNSAT_i$ and hence B_i when setting x_i to *True* and when setting x_i to *False*. The algorithm's plan is to randomly set variable x_i so as to increase $\mathbb{E}[B_i - B_{i-1}]$ where the expectation is conditioned on the setting of the propositional variables $x_1, \ldots, x_{i-1}$ in the previous iterations.

To that end, let t_i (resp. f_i) be the value of $B_i - B_{i-1}$ when x_i is set to *True* (resp., *False*). In order to ensure that the value of the partial solution cannot decrease in any iteration, we would first prove $f_i + t_i \geq 0$. If $t_i \geq 0$ and $f_i \leq 0$, we assign $x_i = \mathit{True}$. If $t_i \leq 0$ and $f_i > 0$, we assign $x_i = \mathit{False}$. If both $t_i > 0$ and $f_i > 0$ then we assign x_i proportionally; that is, $x_i = \mathit{True}$ with probability $\frac{t_i}{t_i + f_i}$ and set False otherwise. Thus, we obtain the randomized $\frac{3}{4}$ online algorithm.

De-randomnizing the $\frac{3}{4}$ competitive Max-Sat algorithm

Polyczek et al also showed how the $\frac{3}{4}$ algorithm can be "de-randomized" into a two pass online algorithm by the method of conditional expectations.

The basic idea is to use a first pass to create a probability p_i for determinining the truth value of the i^{th} propositional variable x_i, and then use the method of conditional expectations in a second pass to determninistically set the truth value for x_i.

However, in terms of the randomized computation tree we are trying to de-randomize, we want to have the same probability for every node at the i^{th} level of the randomized computation tree; that is, not dependent on the path to that node. In other words, we want to be able to derandomize based on p_i and the previous values given to $x_1, \ldots, x_{i-1}$. This can be accomplished by a modification of the analysis of the randomized algorithm.

Can priority order or random order provide improved ratios?

Experimental results for MaxSat algorithms

Table I. The Performance of Greedy Algorithms

	RG		JA		2Pass	
	% Sat	∅ Time	% Sat	∅ Time	% Sat	∅ Time
SC-APP	97.42	0.38s	98.71	0.38s	99.48	1.11s
MS-APP	95.69	0.25s	97.97	0.23s	98.08	0.63s
SC-CRAFTED	97.40	0.17s	98.37	0.17s	99.04	0.46s
MS-CRAFTED	80.33	0.00s	82.69	0.00s	82.97	0.00s
SC-RANDOM	97.58	1.39s	98.72	1.38s	99.19	5.38s
MS-RANDOM	84.61	0.00s	87.30	0.00s	88.09	0.00s

Figure: Table taken from Poloczek and Williamson [2017]. RG is the Poloczek and Williamson algorithm, JA is Johnson's algorithm

Table II. The Performance of Local Search Methods

	NOLS+TS		2Pass+NOLS		SA	
	% Sat	∅ Time	% Sat	∅ Time	% Sat	∅ Time
SC-APP	90.53	93.59s	99.54	45.14s	99.77	104.88s
MS-APP	83.60	120.14s	98.24	82.68s	99.39	120.36s
SC-CRAFTED	92.56	61.07s	99.07	22.65s	99.72	70.07s
MS-CRAFTED	84.18	0.65s	83.47	0.01s	85.12	0.47s
SC-RANDOM	97.68	41.51s	99.25	40.68s	99.81	52.14s
MS-RANDOM	88.24	0.49s	88.18	0.00s	88.96	0.02s

Figure: Table taken from Poloczek and Williamson [2017]. NOLS+TS is non-oblvious local search initialized by Tabu Search, SA is simulated annealing