CSC2421 Topics in Algorithms: Online and Other Myopic Algorithms Fall 2019

Allan Borodin

October 16, 2019

Announcement: Friday 11AM, October 25, MP137

Spyros Angelopolous: "Online Computation with Untrusted Advice"
The advice model of online computation captures the setting in which an online algorithm is given some partial information concerning the request sequence. This paradigm allows us to establish tradeoffs between the amount of this additional information and the performance of the online algorithm. However, unlike real life in which advice is a recommendation that we can chose to follow or to ignore based on trustworthiness, in the current advice model, the online algorithm typically treats it as infallible. This means that if the advice is corrupt or, worse, if it comes from a malicious source, the algorithm may perform poorly. In this work, we study online computation in a setting in which the advice is provided by an untrusted source. Our objective is to quantify the impact of untrusted advice so as to design and analyze robust online algorithms that are resilient and perform well even when the advice is generated in a malicious, adversarial manner. We show how the new paradigm can be applied to well-studied online problems such as ski rental, online bidding, bin packing, and list update.

Todays agenda

- A quick summary of graph coloring results. This will appear in the clides for W4 but I am temporarily incuding them in todays slides since we were interupted last time by an alarm.
- Chapter 6 topics: Max-Sat and Maximizing an unconstrained non-monotone submodular function
- De-randomization by the method of conditional expectations
- The randomized $\frac{3}{4}$ competitive algorithm for Max-SAT
- The Buchbinder et al "two-sided online algorithm" for the maximizing an unconstrained non-monotone submodular function
- The "natural" randomization of some deterministic algorithms.
- Wishful thinking: Possibly start discussion of primal based ual online algorithms.

Online graph coloring

Given a graph $G = (V, E)$, a function $c : V \to [k]$ is called a valid k-coloring of G if for every edge $\{u, v\} \in E$ we have $c(u) \neq c(v)$. The value $c(v)$ is called a color of the vertex v. A graph is called k-colorable if there exists a valid k-coloring.

Trivially, with the exception of the complete graph which requires n colors, every graph on $n = |V|$ vertices is $(n-1)$-colorable. However, many graphs can be colored with many fewer colors. Constructing a valid coloring online in the VAM-PH input model is difficult for general graphs. More precisely, we have the following result:

Theorem

For any deterministic online algorithm ALG for coloring in the VAM-PH input model, there exists a $\log n$*-colorable graph* G *such that ALG uses at least* $2n/(\log n)$ *colors. In other words, we have* $\rho(ALG) \geq \frac{2n}{\log^2(n)}$.

Coloring k-colorable graphs

The theorem shows that the class of $\log n$-colorable graphs does not admit online algorithms that are significantly better than the trivial algorithm. For the case of constant $k \geq 3$ colors, the competitive and approximation ratios remain an open question. Blum and Karger's (1997) $\tilde{O}(n^{3/14})$ approximation remains the best known offline approximation for 3-colorable graphs. (Coloring 3-colorable graphs is an NP-hard problem.)

The current state of the art for the online competitive ratio is $O(n^{1-\frac{1}{k!}})$ for coloring k-colorable graphs (and, additionally an improved $\tilde{O}(n^{2/3})$ for 3-coloroable graphs) due to Kierstead (1998).

For the case of $k = 2$ (i.e., bipartite graphs), tight bounds are known. We first note that trees are a special case of 2-colorable graphs. The following result shows that for every deterministic coloring algorithm ALG, there is a tree T with 2^{k-1} nodes such ALG uses k colors in coloring T. This will show that the competitive ratio $\rho \geq \frac{\log n}{2}$. A somewhat more involved argument shows that $\rho \geq \log n$ for which there is a corresponding positive result.

A lower bound for coloring trees

Theorem

Let ALG be a deterministic online algorithm for Graph Coloring problem restricted to bipartite graphs in the VAM-PH input model. Then there is a tree T with $n-1$ nodes such ALG uses atv least $\log n$ *colors when coloring T. That is, $\rho(ALG) \geq \frac{\log n}{2}$.*

Proof.

We prove the following statement by induction on k: given an arbitrary sequence of input items $I_1, \ldots, I_m$, the adversary can extend the sequence with disjoint trees $T_1, T_2, T_3, \ldots$ such that ALG colors roots of the trees with k different colors and the combined size of the trees is $\leq 2^k - 1$. See text.

First Fit is an optimal coloring algorithm for trees

The natural greedy coloring algorithm is First Fit. That is, for each new node, color it with the smallest non-conflicting color.

Theorem

For online coloring of trees, First Fit achieves the optimaal (except for the +1) $\rho = \log n + 1$ competitive ratio.

Coloring bipartite graphs

Since First Fit is an optimal online coloring algorithm for trees, it is natural to ask how well First Fit performs on the more general class of bipartite graphs. It is easy to see that First Fit can do very poorly when coloring some bipartite graphs.

Fact

$\rho(FirstFit) \geq n/4$.

Proof.

Let $G_n = (U, V, E)$ be the $2n$ node bipartite graph where $|U| = |V| = n$ and $E = \{(u_i, v_j) | i \neq j\}$. That is G_n is a complete bipartite graph minus the edges $\{u_i, v_i\}$. The adversary presents the vertices in the following order: $u_1, v_1, u_2, v_2, \ldots u_n, v_n\}$. First Fit wll use n colors on this input sequence compared to the optimal 2 colors.

□

An optimal online coloring algorithm for bipartite graphs

While First Fit performs poorly on bipartite graphs, there is an online algorithm that will color every bipartite graph with n colors. Rather than greedily coloring each node v, the algorithm just avoids using the same color in the connected component in which v initially occurs. Here is the algorithm:
When a vertex v arrives, CBIP computes the connected component C_v (so far) to which v belongs. Since the entire graph is bipartite, C_v is also bipartite. CBIP computes a partition of C_v into two blocks: S_v that contains v and $\widetilde{S}_v$ that does not contain v. In other words, $C_v = S_v \cup \widetilde{S}_v$. Note that neighbors of v are only among $\widetilde{S}_v$. Let i denote the smallest color that does not appear in $\widetilde{S}_v$. CBIP colors v with color i.

Theorem

For coloring bipartite graphs, we have

$$\rho(CBIP) \leq \log n.$$

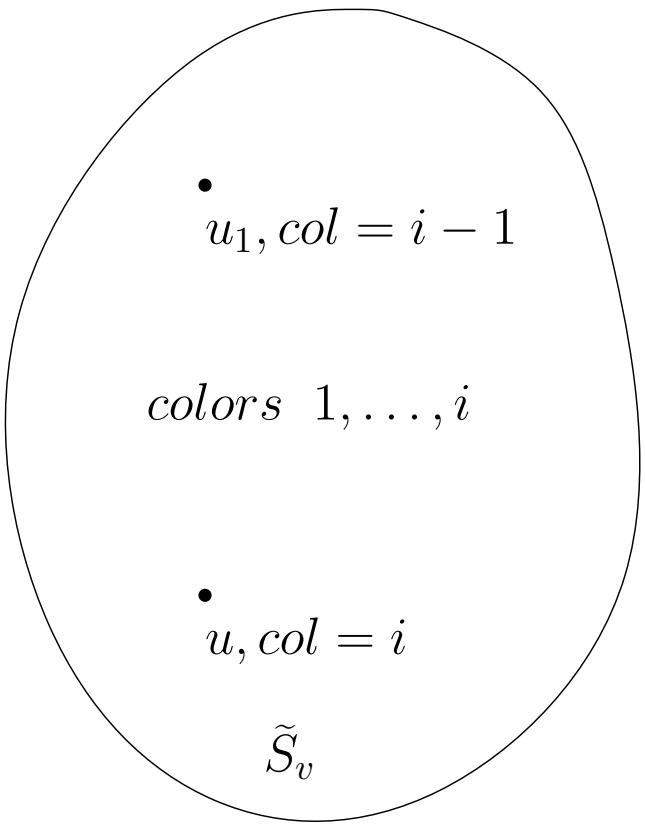
Proving the $\log n$ competitive ratio for coloring bipartite graphs

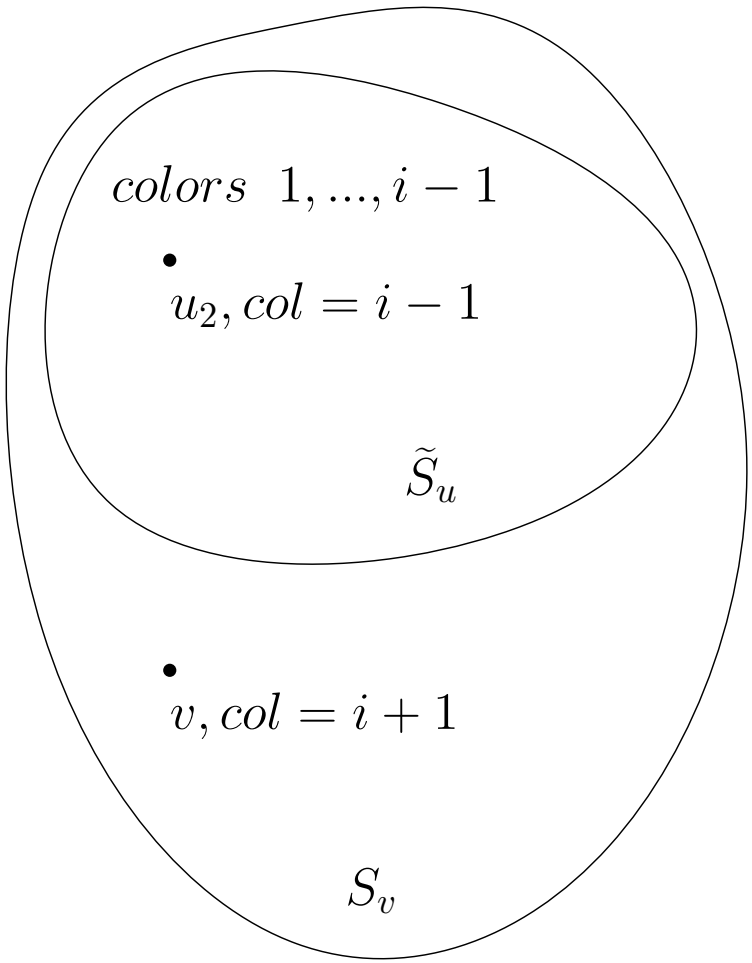
Let $n(i)$ denote the minimum number of nodes that have to be presented to CBIP in order to force it to use color i for the first time. We want to show that $n(i) \geq \lceil 2^{i/2} \rceil$ by induction on i.
Base cases: clearly we have $n(1) = 1$ and $n(2) = 2$.

Inductive step: let v be the first vertex that is colored with color $i+1$ by CBIP. Consider C_v, S_v, and $\widetilde{S}_v$ as defined.In particular, *all colors* $1, 2, \ldots, i$ appear among $\widetilde{S}_v$. Let u be a vertex in $\widetilde{S}_v$ that is colored i. Let $C_u, S_u, \widetilde{S}_u$ be defined for the vertex u at the time that it appeared. Since u was assigned color i, then all colors $1, 2, \ldots, i-1$ appeared in $\widetilde{S}_u$. Observe that $\widetilde{S}_u \subseteq S_v$. Therefore, there exists vertex $u_1 \in \widetilde{S}_v$ colored $i-1$ and there exists vertex $u_2 \in S_v$ colored $i-1$, as well.
Without loss of generality assume that $u_1 \prec u_2$. At the time that u_2 was colored, the connected component of u_2 had to be disjoint from the connected component of u_1, for otherwise u_2 would not have been colored with the same color as u_1.

Finishing the proof for online bipartite graph coloring

Thus, we have $C_{u_1} \cap C_{u_2} = \varnothing$. Furthermore, we can apply the inductive assumption to each of C_{u_1} and C_{u_2} to get that $|C_{u_1}|, |C_{u_2}| \geq \lceil 2^{(i-1)/2} \rceil$. Hence the number of vertices that have been presented prior to v is at least $|C_{u_1}| + |C_{u_2}| \geq 2\lceil 2^{(i-1)/2} \rceil \geq \lceil 2^{(i+1)/2} \rceil$.





What other classes of graphs have small competitive ratios for coloring? A tour of some graph classes

Since trees are such a constrained class of graphs, there are many generalizations. One of the most studied generalizations is that of *bounded tree width graphs*. Tree width graphs are further generalized by the class of (degree) d-inductive graphs.

A graph G is called *d-inductive* (also called d-degenerate) if there is an ordering of vertices such that every node u has at most d neighbors appearing after u in the ordering. When a d-inductive graph G is given in the VAM-PH input model, the order in which the nodes are presented is adversarial and can be different from an inductive ordering.

Continued tour of some graph classes and their online coloring

If the nodes of G are presented in the reverse order of an inductive order then the FirstFit algorithm uses at most $d + 1$ colors. This implies that every d-inductive graph is $(d + 1)$-colorable. We show in the text that when a d-inductive graph is given in an adversarial VAM-PH model, the FirstFit algorithm uses at most $O(d \log n)$ colors. In particular, this generalizes the fact that FirstFit achieves competitive ratio $\log n$ on trees, noting that trees are 1-inductive.

Theorem

FirstFit uses at most $O(d \log n)$ colors on any d-inductive graph G under the VAM-PH input model.

Continued tour of graph classes

An interval graph is the graph induced by the intersection of intervals on the line. Interval graphs are a special case of *chordal graphs* which in turn are *perfect graphs*. There is a characterization of chordal graphs using the concept of a *perfect elimnination ordering*. For a graph G, an order $v_1, v_2, \ldots, v_n$ is a prefect elimination order if for all i, $Nbhd(v_i) \cap \{v_i + 1, \ldots, v_n\}$ is a clique. Equivalently $Nbhd(v_i) \cap \{v_{i+1}, \ldots, v_n\}$ does not have 2 independent (i.e. non adjacent) nodes.

For any graph G, we have $\chi(G) \geq \omega(G)$ where $\chi(G)$ is the chromatic number of a graph and $\omega(G)$ is the clique number of a graph. A perfect graph G is one that satisfies $\chi(G) = \omega(G)$.

We can generalize chordal graphs to the class of d-inductively independent graphs which have a vertex ordering $v_1, v_2, \ldots, v_n$ such that $Nbhd(v_i) \cap \{v_i + 1, \ldots, v_n\}$ does not have $d + 1$ independent nodes.

Clearly Every d-indpendent graph is d-inductively independent.

Online coloring of interval graphs

For interval graphs, it is known that First Fit has a constant (but not optimal) competitive ratio. However, we have the following:

Theorem

For every deterministic online coloring algorithm ALG for interval graphs, there is an interval graph such that ALG uses $3\omega - 2$ colours. There is a deterministic online coloring algorithm ALG^* that colors every interval graph G using at most $3\omega(G) - 2$ colors. Therefore, $\rho(ALG^*) = 3$.

Albers and Schraink (2017) provide randomized online coloring lower bounds for many graphs classes including trees, chordal graphs, and d-inductive graphs. The tree lower bound also implies that planar and bipartite graphs also have an $\Omega(\log n)$ lower bound on the randomized competitive ratio.

Sketch of the interval coloring algorithm

The algorithm might be called *online recursive greedy*
For a sequence of initial nodes in the interval graph, the congestion number (i.e., clique number) is the maximum number of edge intersections. For each clique number k, we can recursively define an algorithm $RECG_k$:

The base case trivially colors all nodes witn color 1.

Induction step assuming $RECG_k$ has been defined. Consider the sequence $\sigma = v_1, \ldots v_i$ which we assume has clique number at most $k+1$. Let $A = \{v_{j_1}, \ldots, v_{j_r}\}$ be a minimal set of nodes (i.e., intervals) such that $\sigma' = \sigma \setminus A$ can be colored with k colors.
If $\sigma' v_{i+1}$ has cliques number k, then color v_{i+1} using $RECG_k$. Otherwise use color $k+1$ for v_{i+1}.

The online interval coloring algorithm

The online algorithm uses $RECG_k$ until clique number $k+1$ is created by some new vertex v_i and then we start using $RECG_{k+1}$.

The proof of the $3\omega - 2$ competitive ratio relies on showing that Av_i can be colored using only 3 new colors.

The ends our discussion of Chapter 5.

Chapter 6 concerns two perhaps seemingly unrelated problems:

1. Maximizing a non-monotone submodular function (without any constraints)
2. Max-Sat

Max-Sat

Max-Sat is a natural optimization problem relating to the original *NP*-complete problem SAT. It is arguably the most studiend constraint satisfaction problem.

In SAT (or more descriptive CNF-SAT), we are given a formula in CNF form; that is $F = C_1 \wedge C_2 \ldots \wedge C_m$ where each C_j is a *clause*, which is a disjunction of literals, where a literal is a propositional variable x_i or its negation $\bar{x}_i$. For example: $C = x_1 \vee \bar{x}_2 \vee x_3$. The objective is to decide whether or not the input formul F is satisfiable.

In Max-Sat, we are again given a CNF formula F, and we want to obtain a truth assignment τ so as to maximize the number of clauses satisfied by τ. If the clauses are weighted then we want to obtain a truth assignment so as to maximize the total weight of satisfied clauses.

Online Max-Sat

In the online version of this problem, input items are

(variable names x_i, and information about clauses where x_i appears).

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

- (Input model 0): For each x_i only the names of the clauses in which x_i occurs positively and those in which it 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.

Max-k-Sat and Exact Max-k-Sat

- Consider the exact Max-k-Sat problem where we are given a CNF propositional formula in which every clause has exactly k literals. We consider the weighted case in which clauses have weights. The goal is to find a satisfying assignment that maximizes the size (or weight) of clauses that are satisfied.
- As already noted, since exact Max-k-Sat generalizes the exact k- SAT decision problem, it is clearly an NP hard problem for $k \geq 3$. It is interesting to note that while 2-SAT is polynomial time computable, Max-2-Sat is still NP hard.
- The naive randomized (online) algorithm for Max-k-Sat is to randomly set each variable to *true* or *false* with equal probability. **Note:** This is an online algorithm. We can think of each propositional variable arriving online and then setting its value randomily.

Analysis of naive Max-k-Sat algorithm continued

- Since the expectation of a sum is the sum of the expectations, we just have to consider the probability that a clause is satisfied to determine the expected weight of a clause.
- Since each clause C_i has k variables, the probability that a random assignment of the literals in C_i will set the clause to be satisfied is exactly $\frac{2^k-1}{2^k}$. Hence $\mathbf{E}$ [weight of satisfied clauses] $= \frac{2^k-1}{2^k} \sum_i w_i$
- Of course, this probability only improves if some clauses have more than k literals. It is the small clauses that are the limiting factor in this analysis.
- This is not only an approxination ratio but moreover a "totality ratio" in that the algorithms expected value is a factor $\frac{2^k-1}{2^k}$ of the sum of all clause weights whether satisfied or not.
- We can hope that when measuring against an optimal solution (and not the sum of all clause weights), small clauses might not be as problematic as they are in the above analysis of the naive algorithm.

Derandomizing the naive algorithm

We can derandomize the naive algorithm by what is called the method of conditional expectations. Let $F[x_1, \ldots, x_n]$ be an exact k CNF formula over n propositional variables $\{x_i\}$. For notational simplicity let $true = 1$ and $false = 0$ and let $w(F)|\tau$ denote the weighted sum of satisfied clauses given truth assignment τ.

- Let x_j be any variable. We express $\mathbf{E}[w(F)|_{x_i \in_u \{0,1\}}]$ as $\mathbf{E}[w(F)|_{x_i \in_u \{0,1\}} | x_j = 1] \cdot (1/2) + \mathbf{E}[w(F)|_{x_i \in_u \{0,1\}} | x_j = 0] \cdot (1/2)$
- This implies that one of the choices for x_j will yield an expectation at least as large as the overall expectation.
- It is easy to determine how to set x_j since we can calculate the expectation clause by clause.
- We can continue to do this for each variable and thus obtain a deterministic solution whose weight is at least the overall expected value of the naive randomized algorithm.
- NOTE: The derandomization can be done so as to achieve an online algorithm. Here the (online) input items are the propostional variables. What input representation is needed/sufficient?

(Exact) Max-k-Sat

- For exact Max-2-Sat (resp. exact Max-3-Sat), the approximation (and totality) ratio is $\frac{3}{4}$ (resp. $\frac{7}{8}$).
- For $k \geq 3$, using PCPs (probabilistically checkable proofs), Hastad proves that it is NP-hard to improve upon the $\frac{2^k-1}{2^k}$ approximation ratio for Max-k-Sat.
- For Max-2-Sat, the $\frac{3}{4}$ ratio can be improved by the use of semi-definite programming (SDP) and randomized rounding.
- The analysis for exact Max-k-Sat clearly needed the fact that all clauses have at least k clauses. What bound does the naive online randomized algorithm or its derandomztion obtain for (not exact) Max-2-Sat or arbitrary Max-Sat (when there can be unit clauses)? The analysis only guarantees a $\frac{1}{2}$ approximation

Johnson's Max-Sat Algorithm

Johnson's [1974] algorithm

For all clauses C_i, $w_i' := w_i/(2^{|C_i|})$
Let L be the set of clauses in formula F and X the set of variables
For $x \in X$ (or until L empty)
  Let $P = \{C_i \in L$ such that x occurs positively$\}$
  Let $N = \{C_j \in L$ such that x occurs negatively$\}$
  If $\sum_{C_i \in P} w_i' \geq \sum_{C_j \in N} w_j'$
    $x := true; L := L \setminus P$
    For all $C_r \in N, \quad w_r' := 2w_r'$ **End For**
  Else
    $x := false; L := L \setminus N$
    For all $C_r \in P, \quad w_r' := 2w_r'$ **End For**
  End If
  Delete x from X
End For

Aside: This reminds me of boosting (Freund and Shapire [1997])

Johnson's algorithm is the derandomized algorithm

- Twenty years after Johnson's algorithm, Yannakakis [1994] presented the naive algorithm and showed that Johnson's algorithm is the derandomized naive algorithm.
- Yannakakis also observed that for arbitrary Max-Sat, the approximation of Johnson's algorithm is at best $\frac{2}{3}$. For example, consider the 2-CNF $F = (x \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge \bar{y}$ when variable x is first set to true. Otherwise use $F = (x \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge y$.
- Chen, Friesen, Zheng [1999] showed that Johnson's algorithm achieves approximation ratio $\frac{2}{3}$ for arbitrary weighted Max-Sat.
- For arbitrary Max-Sat (resp. Max-2-Sat), the current best approximation ratio is .7968 (resp. .9401) using semi-definite programming and randomized rounding.
  Note: While existing combinatorial algorithms do not come close to these best known ratios, it is still interesting to understand simple and even online algorithms for Max-Sat.

Improving on Johnson's algorithm

- In proving the (2/3) approximation ratio for Johnson's Max-Sat algorithm, Chen et al asked whether or not the ratio could be improved by using a random ordering of the propositional variables. The question asked by Chen et al was answered by Costello, Shapira and Tetali [2011] who showed that in the ROM model, Johnson's algorithm achieves approximation $(2/3 + \epsilon)$ for $\epsilon \approx .003653$
- Poloczek and Schnitger [same SODA 2011 conference] show that the approximation ratio for Johnson's algorithm in the ROM model is at most $2\sqrt{15}–7 \approx .746 < 3/4$, noting that $\frac{3}{4}$ is an offline approxaimtion ratio first obtained by Yannakakis' IP/LP algorithm.
- Poloczek and Schnitger first consider a "canonical randomization" of Johnson's algorithm; namely, the canonical randomization sets a variable $x_i = \mathit{true}$ with probability $\frac{w_i'(P)}{w_i'(P)+w_i'(N)}$ where $w_i'(P)$ (resp. $w_i'(N)$) is the current combined weight of clauses in which x_i occurs positively (resp. negatively). Their substantial additional idea is to adjust the random setting so as to better account for the weight of unit clauses in which a variable occurs.

A few comments on the Poloczek and Schnitger algorithm

- The Poloczek and Schnitger algorithm is called Slack and has approximation ratio $= 3/4$.
- The Slack algorithm is a randomized online algorithm (i.e. adversary chooses the ordering) where the variables are represented within input Model 1.
- This approximation ratio is in contrast to Azar et al [2011] who prove that no randomized online algorithm can achieve approximation better than $2/3$ when the input model is input model 0.
- Finally (in this regard), Poloczek [2011] shows that no deterministic priority algorithm can achieve a $3/4$ approximation within input model 2. This provides a sense in which to claim that the Poloczek and Schnitger Slack algorithm "cannot be derandomized".
- The best deterministic priority algorithm in the third (most powerful) model remains an open problem as does the best randomized priority algorithm and the best ROM algorithm.

Revisiting the "cannot be derandomized comment"

Spoiler alert: we will be discussing how algorithms that cannot be derandomized in one sense can be deramdomized in another sense.

- The Buchbinder et al [2012] online randomized 1/2 approximation algorithm for Unconstrained Submodular Maximization (USM) cannot be derandomized into a "similar" deterministic algorithm by a result of Huang and Borodin [2014].
- However, Buchbinder and Feldman [2016] show how to derandomize the Buchbinder et al algorithm into an algorithm that generates $2n$ parallel streams where each stream is an online algorithn.
- The Buchbinder et al USM algorithm is the basis for a randomized 3/4 approximation online MaxSat (even Submodular Max Sat) algorithm.
- Pena and Borodin show how to derandomize this 3/4 approximation algorithm following the approach of Buchbinder and Feldman.

A two pass de-randomization for Max-Sat

- Poloczek et al [2017] de-randomize a Max-Sat algorithm using a 2-pass online algorithm.
- More specifically, using ideas from Buchbinder and Feldman [2015], it can be shown how to derandomize an algorithm equivalent to the Buchbinder et al Max Sat algorithm into a "polynomial width online algorithm" that essentially adaptively generates the randomization tree in such a way that it limits the support of the distribution.
- This 2-pass algorithm is interesting theoretically as it is a deterministic combinatiorial $\frac{3}{4}$-approximation algorithm that works well "in practice". See experimental paper by Polozek and Williamson [2017]. It is online in the sense that the algorithm processses the propositional variables in the same adversarial order in both passes.

A randomized online $\frac{3}{4}$ competitive ratio for Max-Sat

The following algorithm turns out to be equivalent (when restricted to weighted Max-Sat rather than the submodular Max-Sat) to the Buchbinder et al algorithm that was derived from their unconstrained (non-monotone) submodular maximization algorithm. This algorithm was independently given by van Zuylen. The equivalence with Buchbinder et al and the de-randomization to a two pass algorithm appears in a paper by Poloczek, Schnitger, Williamson and van Zuylen [2017].

The idea of the algorithm is that in setting the variables, we want to balance the weight of clauses satisfied with that of the weight of clauses that are no longer satisfiable.

Let S_i be the assignment to the first i variables and let SAT_i (resp. $UNSAT_i$) be the weight of satisfied clauses (resp., clauses no longer satisfiable) with respect to S_i. Let $B_i = \frac{1}{2}(SAT_i + W - UNSAT_i)$ where W is the total weight of all clauses.

A randomized online $\frac{3}{4}$ competitive ratio for Max-Sat

The following algorithm turns out to be equivalent (when restricted to weighted Max-Sat rather than the submodular Max-Sat) to the Buchbinder et al algorithm that was derived from their unconstrained (non-monotone) submodular maximization algorithm. This algorithm was independently given by van Zuylen. The equivalence with Buchbinder et al and the de-randomization to a two pass algorithm appears in a paper by Poloczek, Schnitger, Williamson and van Zuylen [2017].

The idea of the algorithm is that in setting the variables, we want to balance the weight of clauses satisfied with that of the weight of clauses that are no longer satisfiable.

Let S_i be the assignment to the first i variables and let SAT_i (resp. $UNSAT_i$) be the weight of satisfied clauses (resp., clauses no longer satisfiable) with respect to S_i. Let $B_i = \frac{1}{2}(SAT_i + W - UNSAT_i)$ where W is the total weight of all clauses.

The algorithm's plan is to randomly set variable x_i so as to increase $\mathbb{E}[B_i - B_{i-1}]$.

The randomized max-sat approximation algorithm continued

Let t_i (resp. f_i) be the value of $B_i - B_{i-1}$ when x_i is set to true (resp. false).

For $i = 1 \ldots n$
  If $f_i \leq 0$, then set $x_i =$ true
  Else if $t_i \leq 0$,
    then set $x_i =$ false
  Else set x_i true (resp. false) with probability $\frac{t_i}{t_i+f_i}$ (resp. $\frac{f_i}{t_i+f_i}$).
End For

Note that this is an online algorithm since $SAT_i, UNSAT_i$ can be computed online and hence so can B_i, t_i and f_i.

The analysis of the $\frac{3}{4}$ competitive ratio

Consider an optimal solution (even an LP optimal) $\mathbf{x}^*$ and let OPT_i be the assignment in which the first i variables are as in S_i and the remiaing $n - i$ variables are set as in $\mathbf{x}^*$. (Note: x^* is not calculated.)

We need to show the following main lemma:

- $\mathbb{E}[w(OPT_{i-1}) - w(OPT_i)] \leq \mathbb{E}[w(B_i) - w(B_{i-1})]$

We will first show how the competitive ratio follows from the main lemma above.

We will then proceed to prove the main lemma with the help of some additional lemmas.

Proof of competitive ratio using the main lemma

We first note that the initial assignment is $S_0 = OPT_0 = OPT_{LP}$ and $S_n = OPT_n$ is the final assignment of the algorithm.
Furthermore, $W = \sum_j w_j$, $w(B_0) = \frac{1}{2}W$, and $w(B_n) = w(S_n)$ is the weight of the clauses satisfied by the assignment of the online algorithm.

The proof establishes something stronger than stated. Namely,

$$\mathbb{E}[w(S_n)] \geq \frac{2w(OPT_{LP}) + W}{4} \geq \frac{3}{4}w(OPT_{LP}) \geq \frac{3}{4}w(OPT)$$

Proof of the stronger statement

Summing the inequalities provided by the main lemma, we have

$$\sum_{i=1}^{n} \mathbb{E}[w(OPT_{i-1}) - w(OPT_i)] \leq \sum_{i=1}^{n} \mathbb{E}[B_i - B_{i-1}]$$

This then implies (by linearity of expectations) and telescoping that

$$\mathbb{E}[w(OPT_0)] - w(OPT_n)] \leq \mathbb{E}[B_n] - \mathbb{E}[B_0]$$

Restated:

$$w(OPT_{LP}) - \mathbb{E}[w(S_n)] \leq \mathbb{E}[w(S_n)] - \frac{1}{2}W$$

Rearranging and dividing by 2::

$$\frac{1}{2}(OPT_{LP}) + \frac{1}{4}W \leq \mathbb{E}[w(S_n)]$$

Which (since $OPT_{LP} \leq W$) implies the desired result:

$$\frac{3}{4}w(OPT) \leq \frac{3}{4}w(OPT_{LP}) \leq \frac{1}{2}(OPT_{LP}) + \frac{1}{4}W \leq \mathbb{E}[w(S_n)]$$

The supporting lemmas needed for the main lemma

We need to establish the following lemmas

Lemma 1: $f_i + t_i \geq 0$. Note that this lemma insures that the weight of the partial solution cannot decrease in any iteration.

Lemma 2: If $f_i + t_i > 0$, then
$\mathbb{E}[w(OPT_{i-1} - w(OPT_i)] \leq \max\{0, \frac{2f_i t_i}{f_i + t_i}\}$
Supporting Lemma 2 allows us to prove the main by considering two cases:

Case 1: $f_i \leq 0$ or $t_i \leq 0$ in which case x_i is set deterministically so that $\mathbb{E}[B_i] = \mathbb{E}[B_{i-1}]$. The lemma follows since it assumed $f_i + t_i > 0$ and hence we must have $f_i t_i \leq 0$.

Case 2: $f_i > 0$ and $t_i > 0$ so that $f_i t_i > 0$ and $f_i + t_i > 0$.
By definition

$$\begin{aligned}
\mathbb{E}[B_i - B_{i-1}] &= \frac{f_i}{f_i+t_i}\mathbb{E}[B_i(false) - B_{i-1}] + \frac{t_i}{f_i+t_i}\mathbb{E}[B_i(true) - B_{i-1}] \\
&= \frac{f_i^2+t_i^2}{f_i+t_i} \\
&\geq \frac{2f_i t_i}{f_i+t_i} \qquad \text{since } f_i^2 - 2f_i t_i + t_i^2 = (f_i - t_i)^2 \geq 0 \\
&\geq \mathbb{E}[w(OPT_{i-1} - w(OPT_i)] \qquad \text{by the supporting lemma.}
\end{aligned}$$

Proof of supporting lemma 1

- $SAT_i(true/false)$ = weight of clauses satisfied by the partial assignment S_{i-1} and setting $x_i = true/false$.
- $UNSAT_i(true/false)$ are defined analogously for the weight of unsatisfiable clauses by the partial assignment S_{i-1} and setting $x_i = true/false$.

Therefore
$B_i(true/false) = \frac{1}{2}(SAT_i(true/false) + (W - UNSAT_i(true/false)))$ and $t_i = B_i(true) - B_{i-1}$ and $f_i = B_i(false) - B_{i-1}$.
Observe that if a clause becomes unsatisfied for the first time by setting x_i to be true then that clause becomes satisfied by setting x_i to false (and vice versa). This implies:

$UNSAT_i(true/false) - UNSAT_{i-1} \leq SAT_i(false/true) - SAT_{i-1}$.
Therefore, $\frac{1}{2}(SAT_i(true) - SAT_{i-1}) + \frac{1}{2}(SAT_i(false) - SAT_i) \geq \frac{1}{2}(UNSAT_i(false) - UNSAT_{i-1}) + \frac{1}{2}(UNSAT_i(true) - UNSAT_{i-1})$. With some rearranging of this inequality we get the desired $t_i + f_i \geq 0$

Proof of supporting lemma 2

We need to bound the change in $w(OPT_i)$ when the algorithm replaces x_i^* by $x_i = \mathit{true}$. (An analogous argument can be made when when the algorithm replaces x_i^* by $x_i = \mathit{false}$.) Equivalently, lets consider $w(OPT_{i-1}) - w(OPT_i)$ when we reverse setting $x_i = \mathit{true} = 1$ back to x_i^*. Any clause C_j that contributes positively to $w(OPT_{i-1}) - w(OPT_i)$ must be a clause in which $\bar{x}_i$ occurs and the increase due to such a clause C_j is bounded by $(1 - x_i^*)$ times the weight of C_j. That is, the total possible increase is bounded by $(1 - x_i^*) \cdot [w(SAT_i(\mathit{false})) - w(SAT_{i-1})]$. On the

other hand, any clause C_j containing x_i will cause a decrease of exactly $(i - x_i^*)$ times the weight of C_j. That is, the total decrease is $(1 - x_i^*) \cdot [w(UNSAT_i(\mathit{false})) - w(UNSAT_{i-1})]$. Hence the total change in

$w(OPT_{i-1} - w(OPT_i)$ is at most
$(1 - x_i^*) \cdot [(w(SAT_i(\mathit{false})) - w(Sat_{i-1}) - w(UNSAT_i(\mathit{false})) -$
$w(UNSAT_{i-1})] = (1 - x_i^*) \cdot 2f_i$.

Finishing the proof of supporting lemma 2

By the same type of reasoning, when setting of $x_i = \mathit{false}$, we obtain $w(OPT_{i-1}) - w(OPT_i) \leq x_i^* \cdot 2t_i$.

So to conclude the supporting lemma, we just need to recall how the algorithm is probabilistically setting $x_i = \mathit{true}$ (resp. false) with probability $\frac{t_i}{t_i+f_i}$ (resp. with probability $\frac{f_i}{t_i+f_i}$. We then obtain the desired bound

$$\mathbb{E}[w_(OPT_{i-1}) - w(OPT_i)] \leq \frac{t_i}{t_i+f_i} \cdot (1-x_i^*) \cdot 2f_i + \frac{f_i}{t_i+f_i} \cdot x_i^* \cdot 2t_i = \frac{2f_i t_i}{t_i+f_i}$$

.

This concludes all the required proofs and hence the proof that the randomized mMax-Sat algorithm has competitive ratio at least $\frac{3}{4}$.

A matching $\frac{3}{4}$ inapproximation result for Max-Sat

We recall input model 1 for Max-Sat where each input item is a propositional variable represented by the clauses it appears positively and negatively, and input model 2 which in addition provides the names (but not the signs) of the variables appearing in those clauses.

We claimed that input model 1 was sufficient to implement the randomized Max-Sat algorithm as an online algorithm. Now we claim that even with respect to input model 2, for any $\epsilon > 0$, no randomized online algorithm can achieve competitive ratio $\frac{3}{4} + \epsilon$.

The inapproximation will apply to exact Max-2-Sat. The idea is to constuct a set of 2-clauses and a random distribution on the order of variable arrivals so that no deterministic algorithm can obtain better than the $\frac{3}{4}$ ratio Iin expectation over the input distribution) and then appeal to the Yao principle to infer the randomized result.

The idea behind the inapproximation

The adversary will present variables $x_1, \ldots, x_n$, where each x_i appears in two clauses of length 2 and where the remaining variable is y: in one x_i appears positively and in the other negatively. In fact, the two clauses will either represent an equivalence to y (given by $x_i \vee \overline{y}$, $\overline{x_i} \vee y$) or an inequivalence to y (given by $x_i \vee y$, $\overline{x_i} \vee \overline{y}$), but the algorithm does not know which is the case(by the limitation of input model 2). If an assignment does not satisfy the (in)equivalence correctly, it will get only one of the two clauses (ie 1/2 of the total).

It remains to create a random distribution of instances where the x_i appear first and then y_j valiables so that with high probability, any deterministic sertting of the x_i variables will be between $n/2 - \epsilon n$ and $n/2 + \epsilon n$ in setting the variables of the (in)equivalence correctly.

It is an interesting question as to whether or not the $\frac{3}{4}$ ratio can be beaten using the most general input model 3 which completely reveals all the clauses in which a variable appears.

A related problem: the unconstrained maximnization of a (non-monotone) submodular set function.

- A set function $f : 2^U \to \Re$ is submodular if $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ for all $S, T \subseteq U$.
- Equivalently, f is submodular if it satisfies decreasing marginal gains; that is, $f(S \cup \{x\}) - f(S) \geq f(T \cup \{x\}) - f(T)$ for all $S \subseteq T \subseteq U$ and $x \in U$
- We will always assume that f is *normalized* in that $f(\varnothing) = 0$ and non-negative.
- Submodular functions arise naturally in many applications and has been a topic of much recent activity.
- Probably the most frequent application of (and papers about) submodular functions is when the function is also monotone (non-decreasing) in that $f(S) \leq f(T)$ for $S \subseteq T$.
- Note that linear functions (also called modular) functions are a special case of monotone submodular functions.

Submodular maximization continued

In the submodular maximization problem, we want to compute S so as to maximize $f(S)$.

- For monotone functions, we are maximizing $f(S)$ subject to some constraint (otherwise just choose $S = U$).
- For the non monotone case, the problem is already interesting in the unconstrained case. Perhaps the most prominent example of such a problem is Max-Cut (and Max-Di-Cut).
- Max-Cut is an NP-hard problem. Using an SDP approach just as we will see for the Max-2-Sat problem yields the approximation ratio $\alpha = \frac{2}{\pi} \min_{\{0 \leq \theta \leq \pi\}} \frac{\theta}{(1-cos(\theta)} \approx .87856$. Assuming UGC, this is optimal.
- For a submodular function, we may be given an explicit representation (when a succinct representation is possible as in Max-Cut) or we access the function by an oracle such as the *value oracle* which given S, outputs the value $f(S)$ and such an oracle call is considered to have $O(1)$ cost. Other oracles are possible (e.g. given S, output the element x of U that maximizes $f(S \cup \{x\}) - f(S)$).

Unconstrained (non monotone) submodular maximization

- Feige, Mirrokni and Vondrak [2007] began the study of approximation algorithms for the unconstrained non monotone submodular maximization (USM) problem establishing several results:
  1. Choosing S uniformly at random provides a $1/4$ approximation.
  2. An oblivious local search algorithm results in a $1/3$ approximation.
  3. A non-oblivious local search algorithm results in a $2/5$ approximation.
  4. Any algorithm using only value oracle calls, must use an exponential number of calls to achieve an approximation $(1/2 + \epsilon)$ for any $\epsilon > 0$.
- The Feige et al paper was followed up by improved local search algorithms by Gharan and Vondrak [2011] and Feldman et al [2012] yielding (respectively) approximation ratios of .41 and .42.
- The $(1/2 + \epsilon)$ inapproximation was augmented by Dobzinski and Vondrak showing the same bound for an explicitly given instance under the assumption that $RP \neq NP$.

The Buchbinder et al (1/3) and (1/2) approximations for USM

In the FOCS [2012] conference, Buchbinder et al gave an elegant linear time deterministic 1/3 approximation and then extend that to a randomized 1/2 approximization. The conceptually simple form of the algorithm is (to me) as interesting as the optimality (subject to the proven inapproximation results) of the result. Let $U = u_1, \ldots u_n$ be the elements of U *in any order*.

The deterministic 1/3 approximation for USM

$X_0 := \emptyset$; $Y_0 := U$
For $i := 1 \ldots n$
  $a_i := f(X_{i-1} \cup \{u_i\}) - f(X_{i-1})$; $b_i := f(Y_{i-1} \setminus \{u_i\}) - f(Y_{i-1})$
  If $a_i \geq b_i$
    then $X_i := X_{i-1} \cup \{u_i\}$; $Y_i := Y_{i-1}$
    else $X_i := X_{i-1}$; $Y_i := Y_{i-1} \setminus \{u_i\}$
  End If
End For

The randomized 1/2 approximation for USM

- Buchbinder et al show that the "natural randomization" of the previous deterministic algorithm achieves approximation ratio $1/2$.
- That is, the algorithm chooses to either add $\{u_i\}$ to X_{i-1} with probability $\frac{a'_i}{a'_i+b'_i}$ or to delete $\{u_i\}$ from Y_{i-1} with probability $\frac{b'_i}{a'_i+b'_i}$ where $a'_i = \max\{a_i, 0\}$ and $b'_i = \max\{b_i, 0\}$.
- If $a_i = b_i = 0$ then add $\{u_i\}$ to X_{i-1}.
- Note: Part of the proof for both the deterministic and randomized algorithms is the fact that $a_i + b_i \geq 0$.
- This fact leads to the main lemma for the deterministic case:

$$f(OPT_{i-1} - f(OPT_i) \leq [f(X_i - f(X_{i-1}] + [f(Y_i) - f(Y_{i-1}]$$

Here $OPT_i = (OPT \cup \{X_i\}) \cap Y_i$ so that OPT_i coincides with X_i and Y_i for elements $1, \ldots i$ and coincides with OPT on elements $i+1, \ldots, n$. Note that $OPT_0 = OPT$ and $OPT_n = X_n = Y_n$. That is, the loss in OPTs value is bounded by the total value increase in the algorithm's solutions.

Applying the algorithmic idea to Max-Sat

Buchbinder et al are able to adapt their randomized algorithm to the Max-Sat problem (and even to the Submodular Max-Sat problem). So assume we have a monotone normalized submodular function f (or just a linear function as in the usual Max-Sat). The adaption to Submodular Max-Sat is as follows:

- Let $\phi : X \to \{0\} \cup \{1\} \cup \varnothing$ be a standard partial truth assignment. That is, each variable is assigned exactly one of two truth values or not assigned.
- Let $\mathcal{C}$ be the set of clauses in formula Ψ. Then the goal is to maximize $f(\mathcal{C}(\phi))$ where $\mathcal{C}(\phi)$ is the sat of formulas satisfied by ϕ.
- An extended assignment is a function $\phi' : X \to 2^{\{0,1\}}$. That is, each variable can be given one, two or no values. (Equivalently $\phi' \subseteq X \times \{0,1\}$ is a relation.) A clause can then be satisfied if it contains a positive literal (resp. negative literal) and the corresponding variable has value $\{1\}$ or $\{0,1\}$ (resp. has value $\{0\}$ or $\{0,1\}$.
- $g(\phi') = f(\mathcal{C}(\phi'))$ is a monotone normalized submodular function. ‘

Buchbinder et al Submodular Max-Sat

Now starting with $X_0 = X \times \varnothing$ and $Y_0 = Y \times \{0, 1\}$, each variable is considered and set to either 0 or to 1 (i.e. a standard assignment of precisely one truth value) depending on the marginals as in USM problem.

Algorithm 3: RandomizedSSAT(f, Ψ)

$X_0 \leftarrow \emptyset$, $Y_0 \leftarrow \mathcal{N} \times \{0, 1\}$.
for i = 1 *to* n **do**
  $a_{i,0} \leftarrow g(X_{i-1} \cup \{u_i, 0\}) - g(X_{i-1})$.
  $a_{i,1} \leftarrow g(X_{i-1} \cup \{u_i, 1\}) - g(X_{i-1})$.
  $b_{i,0} \leftarrow g(Y_{i-1} \setminus \{u_i, 0\}) - g(Y_{i-1})$.
  $b_{i,1} \leftarrow g(Y_{i-1} \setminus \{u_i, 1\}) - g(Y_{i-1})$.
  $s_{i,0} \leftarrow \max\{a_{i,0} + b_{i,1}, 0\}$.
  $s_{i,1} \leftarrow \max\{a_{i,1} + b_{i,0}, 0\}$.
  with probability $s_{i,0}/(s_{i,0} + s_{i,1})^*$ **do**:
  $X_i \leftarrow X_{i-1} \cup \{u_i, 0\}$, $Y_i \leftarrow Y_{i-1} \setminus \{u_i, 1\}$.
  else (with the compliment probability $s_{i,1}/(s_{i,0} + s_{i,1})$) **do**:
  $X_i \leftarrow X_{i-1} \cup \{u_i, 1\}$, $Y_i \leftarrow Y_{i-1} \setminus \{u_i, 0\}$.
return X_n (or equivalently Y_n).

* If $s_{i,0} = s_{i,1} = 0$, we assume $s_{i,0}/(s_{i,0} + s_{i,1}) = 1$.

The primal dual framework

We will temp;orarily skip ovefr chapter 7 and move on to chapter 8. In Chapter 8, we describe a framework for designing and analyzing online algorithms based on linear programming (LP) duality. The design and analysis of algorithms based on the primal-dual correspondence is a fundamental idea that goes well beyond online algorithms.

We begin by recalling some basic definitions from the theory of linear programming in the offline setting. Then we describe how linear programming can be stated in the online setting. The key idea is for the algorithm to maintain two online solutions: one to the primal formulation and another to the dual formulation. The goal of the algorithm is to minimize the gap between the objectives achieved by each of the two online solutions. The competitive ratio can be readily derived from the gap by using approximate complementary slackness. This is called the *primal-dual approach*.

Linear programming

Although linear programs can arise naturally for some problems, our main interest will be with respect to linear programs as a relaxation of integer programs.

For example, perhaps the simplest wys to derive an offline approximation for the weighted vertex cover problem. Let the input be a graph $G = (V, E)$ with a weight function $w : V \to \Re^{\geq 0}$. To simplify notation let the vertices be $\{1, 2, \ldots . n\}$. Then we want to solve the following "natural IP representation" of the problem:

- Minimize $\mathbf{w} \cdot \mathbf{x}$
- subject to $x_i + x_j \geq 1$ for every edge $(v_i, v_j) \in E$
- $x_j \in \{0, 1\}$ for all j.

The *intended meaning* is that $x_j = 1$ iff vertex v_j is in the chosen cover. The constraint forces every edge to be covered by at least one vertex.

Solving an IP is an *NP*-hard problem. Instead we relax the integral $x_j \in \{0, 1\}$ constraints and instead allow fractional solutions $x_j \in [0, 1]$. Here the rounding is quite obvious: round LP solution x_j^* to $\bar{x}_j$ iff

The integrality gap

- For LP relaxations of an IP we can define the integrality gap (for a minimization problem) as $\max_{\mathcal{I}} \frac{IP-OPT}{LP-OPT}$; that is, we take the worst case ratio over all input instances $\mathcal{I}$ of the IP optimum to the LP optimum. (For maximization problems we take the inverse ratio.)
- Note that the integrality gap refers to a particular IP/LP relaxation.
- The same concept of the integrality gap can be applied to other relaxations such as in semi definite programming (SDP).
- It should be clear that the simple IP/LP rounding we just used for the vertex cover problem shows that the integrality gap for the previously given IP/LP formulation is at most 2.
- By considering the complete graph K_n on n nodes, it is also easy to see that this integrality gap is at least $\frac{n-1}{n/2} = 2 - \frac{1}{n}$.

Linear programming continued

A linear program is a maximization or minimization problem with a linear objective function subject to linear non-strict inequality constraints. We will first consider minimization problems. Every linear program has an equivalent formulation in the standard form which is the following program P for a minimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i & & \\ \text{subj. to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & & i \in [m] \\ & x_i \geq 0 & & i \in [n] \end{aligned}$$

In the above linear program we have n variables $x_1, \ldots, x_n$ and m constraints involving a_{ij} and b_i constants, as well as n nonnegativity constraints $x_i \geq 0$.

Linear programming continued

We can write down the above program in matrix form as follows:

$$\begin{aligned} \text{minimize} \quad & c^t x \\ \text{subj. to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

In the above, $A = (a_{ij})$ is the matrix of coefficients, $x = (x_1, \ldots, x_n)$, $b = (b_1, \ldots, b_m)$ and $c = (c_1, \ldots, c_n)$ are vectors[1] of variables, right-hand side constant terms, and coefficients of the objective function, respectively. For two vectors v and u of the same dimension, the notation $v \geq u$ is interpreted component-wise, i.e., $v_i \geq u_i$ for all i.

[1] All vectors are column vectors unless stated otherwise. Note that we represent contents of all vectors as row vectors inside text for typographical reasons.

And more linear programming

Any vector $x \in \mathbb{R}^n$ satisfying the constraints of a given LP, i.e., $Ax \geq b$, is called **feasible**. A feasible vector x^* that optimizes the value of the objective is called an **optimal solution**, or simply, an **optimum**.

A linear program might not admit any feasible vectors: a simple example is when there are clearly contradictory constraints such as $x_1 - x_2 \geq 1$, $x_2 - x_1 \geq 1$, and $x_1, x_2 \geq 0$. An LP that does not admit any feasible vectors is called **infeasible**.

When a linear program does have feasible vectors it can be either **bounded**, i.e., has a finite optimum, or **unbounded**, i.e., has an infinite optimum. A simple example of an unbounded LP is to minimize $-x_1$ subject to $x_1 \geq 0$. We shall denote the value of an optimal solution of a linear program P by $OPT(P)$.

Duality

One way of proving lower bounds on the value of $OPT(P)$ is to take non-negative combinations of constraints of P. Let A_i denote the i^{th} row of matrix A. Then the i^{th} constraint of P is $A_i x \geq b_i$. Consider multiplying constraint i by $y_i \geq 0$ and adding up all the resulting inequalities. Then you get $\sum_{i=1}^{m} y_i A_i x \geq \sum_{i=1}^{m} b_i y_i$. In matrix form we can write it as $(A^t y)^t x \geq b^t y$. If we choose y so that $(A^t y) \leq c$ then we get $c^t x \geq (A^t y)^t x \geq b^t y$. Thus, such a choice of y demonstrates a lower bound $b^t y$ on the value of P for *any* feasible x. What is the best possible lower bound that can be derived this way? This is given by another linear program, the **dual** program D:

$$\begin{aligned} \text{maximize} \quad & b^t y \\ \text{subj. to} \quad & A^t y \leq c \\ & y \geq 0 \end{aligned}$$

Relating the primal and the dual

The original linear program P is referred to as **primal**, and (as we just said) the derived linear program D is referred to as **dual**. We have just derived the basic fact about the relationship between the primal and the dual:

Theorem (Weak Duality for a Minimization Problem)

$$OPT(D) \leq OPT(P).$$

We also state the stronger version of this theorem without proof. The proof of the following Strong Duality Theorem relies on more advanced machinery from polytope theory and can be found in any standard text on linear programming.

Theorem (Strong Duality)

If the primal P is feasible and bounded then so is its dual D and, moreover, we have

$$OPT(D) = OPT(P).$$

Approximate complementary slackness

Using linear programming formulations, one of the main tools for proving offline approximation ratios and online competitive ratios is the following approximate complementary slackness theorem.

Theorem (Approximate Complementary Slackness)

Suppose that x is a feasible solution to the primal P and y is a feasible solution to the dual D. If there exist $\alpha, \beta > 1$ such that

- *if $x_i > 0$ then $(c_i/\alpha) \leq (A^t)_i y \leq c_i$; and*
- *if $y_i > 0$ then $b_i \leq A_i x \leq \beta b_i$*

then $c^t x \leq (\alpha\beta) b^t y$.

Proof of complememtary slackness theorem

Proof.

The proof is rather straightforward:

$$\begin{aligned} c^t x &= \sum_{i:x_i>0} c_i x_i \leq \sum_{i:x_i>0} (\alpha (A^t)_i y) x_i \\ &= \sum_{i:x_i>0} \alpha \sum_{j:y_j>0} A_{ji} y_j x_i = \sum_{j:y_j>0} (\alpha y_j) A_j x \\ &\leq \sum_{j:y_j>0} (\alpha y_j)(\beta b_j) = (\alpha\beta) b^t y. \end{aligned}$$

Applying approximate complementary slackness

Approximate complementary slackness can be used to design competitive online algorithms. Suppose that an algorithm maintains a primal solution x and a dual solution y, such that they satisfy the approximate complementary slackness conditions with parameters α and β. Then the solution x is immediately $(\alpha\beta)$ competitive, since

$$OPT(P) = c^t x^* \leq c^t x \leq (\alpha\beta) b^t y \leq (\alpha\beta) b^t y^* = (\alpha\beta) OPT(D),$$

where P is the primal, D is the dual, x^* is an integral optimum for P, and y^* is an LP optimum for D.

There is often a very elegant and general way to uitlize the primal dual framework. We use the primal dual approach to construct a fractional slolution to a problem. We then use the fractional solution to derive a randomized algorithm for the (integral) problem. Finally, we try to de-randomize the algorithm. We will first demonstrate the framework for the online set cover problem.

The offline and online set cover problem

In the offline Set Cover problem, a universe of elements X and a family $\mathcal{S} \subseteq 2^X$ of subsets of X are given to an online algorithm in advance. Moreover, each set $S \in \mathcal{S}$ is associated with a cost $c_S \in \mathbb{R}$, which is also known in advance. We assume that $c_S \geq 1$ for all S. The goal is to select sets from the collection $\mathcal{S}$ to cover X while minimizing the total cost.

It is known that the "natural greedy algorithm" achieves an H_n approximation for the offline set cover problem and it is NP-hard to achieve an approximation better than $\Omega(\log n)$ where $n = |X|$. Using a stronger complexity assumption it is hard to approximate better than H_n.

In the online set cover problem, a target set $X' \subseteq X$ arrives online one element at a time in an adversarial order. The goal is to select sets from the collection $\mathcal{S}$ to cover X' while minimizing the total cost. That is the goal is to find $\mathcal{S}' \subseteq \mathcal{S}$ such that $X' \subseteq \bigcup_{S \in \mathcal{S}'} S$ and $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c_S$ is as small as possible.

The online set cover problem continued

An online algorithm maintains a solution $\mathcal{S}'$ that is initially empty. When an element $e \in X'$ is revealed if it is already "covered" by one of the sets in the partial solution $\mathcal{S}'$ then the online algorithm doesn't have to do anything and the next element from X' can be revealed. If e is not yet covered by one of the sets in $\mathcal{S}'$ then the algorithm has to pick a new set $S \in \mathcal{S}'$ such that $e \in S$ and add that set S to the partial solution $\mathcal{S}'$. This way, the algorithm maintains the invariant that each of the revealed elements is covered by at least one set from $\mathcal{S}'$ at all times. We shall denote the size of X by n and (as stated) the size of $\mathcal{S}$ by m.

Primal dual statement for set cover

To state the primal-dual formulation for Online Set Cover, we introduce primal variables x_S for $S \in \mathcal{S}$ with the intended meaning "$x_S = 1$" indicating that S is part of the solution, and "$x_S = 0$" indicating that S is not part of the solution. In the relaxed LP formulation, the variables x_S can take on fractional values. We denote the dual variables by y_e for $e \in X$. The primal and dual LPs are stated below.

Primal

$$\begin{aligned} \text{minimize} \quad & \sum_{S \in \mathcal{S}} c_S x_S \\ \text{subject to} \quad & \sum_{S \in \mathcal{S}: e \in S} x_S \geq 1 \qquad e \in X' \\ & x_S \in [0,1] \qquad S \in \mathcal{S} \end{aligned}$$

Dual

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in X'} y_e \\ \text{subject to} \quad & \sum_{e \in S} y_e \leq c_S \qquad S \in \mathcal{S} \\ & y_e \geq 0 \qquad e \in X' \end{aligned}$$

The fractional primal dual set cover as an online problem

The online version of Set Cover translates into the following online version of the above primal-dual formulation as follows:

Primal online input: constraints $\sum_{S \in \mathcal{S}: e \in S} x_S \geq 1$ arrive one at a time in the primal; and

Dual online input: variables y_e arriving one at a time in the dual; and

Online decisions: an online algorithm for primal-dual is allowed to update values of variables x_S, y_e only in a monotonically increasing fashion.

A deterministic algorithm for the set cover primal dual formulation

The primal dual formulation just given will lead to a deterministic algorithm that computes a fractional set cover solution with a "good" competitive ratia. Then, as previously foretold, we will be able to transform the fractional algorithm into an integral randomized algorithm with a somewhat worse competitive ratio. And finally we will be able to de-randomize the algorithm maintaining the competitive ratio of the randomized algorithm.

We need some additional notation. Let F_e denote the frequency of element e among sets in $\mathcal{S}$, i.e., $F_e := |\{S \in \mathcal{S} : e \in S\}|$. Let F denote the maximum frequency among input elements, i.e., $F = \max_{e \in X'} F_e$. Clearly $F \leq m$.

The fractional algorithm

All primal variables are initially set to 0
While new elements $e \in X'$ arrive
  A new constraint $\sum_{S \in \mathcal{S}: e \in S} x_S \geq 1$ for the primal arrives.
  A new variable y_e for the dual is introduced.
  While $\sum_{S \in \mathcal{S}: e \in S} x_S < 1$
    For $S \in \mathcal{S}$ such that $e \in S$
      $x_S \leftarrow x_S(1 + 1/c_S) + 1/(c_S F_e)$
    EndFor
    $y_e \leftarrow y_e + 1$
  EndWhile
EndWhile

Theorem

The Algorithm above produces a feasible primal fractional solution. The algorithm yields an integral dual solution that can be scaled down by a factor $O(\log F) = O(\log m)$ to obtain a feasible dual fractional solution. This then establishes that the primal fractional is $O(\log F) = O(\log m)$ competitive.