CSC2421: Online and Other Myopic Algorithms
Fall 2025

Allan Borodin

October 15, 2025

Week 7

Annoucements

- The first problem set is due October 22 at noon.
- Please see or email me about a possible project. Even if we have discussed a project, I would like to have a one paragraph proposal for each project.

Todays agenda

- A few more suggested projects.
- Discussion of the RobustMatch algorithm.
- Online graph problems and their representation.
- Online graph colouring.

A few more possible topic suggestions

Note: I emphasize a project need not obtain new results but should motivate a topic and present a coherent understanding of the topic as to what has been done, open probelms and possible directions for future work.

- In preparing todays class, I thought of some additional possible projects with repect to graph problems. Graph problems seem endless since there are so many graph classes one can consider, different optimization problems, and different myopic models.
- Moving on to later chapters, chapter 12 is only a brief introduction to stochastic inputs. One can look at any online result to see how much better a ratio one can obtain when assuming (known and unkown) i.i.d. input sequences. What is much less studied is when input items are not independent such as in Markov Processes.

More addditional project suggestions

- The recent interest in advice models is about predictions where the objetctive is to obtain good consistency (i.e. performance when the prediction has no or little error) and robustness (i.e., when the preforance is not that bad when the prediction is arbitrarily bad).

  This is now a well studied topic but one can always find problem areas where such prediction advice has either not been studied or where more realistic predictions can be introduced. And as I already suggested, experiments for algorithms with predictions can be very interesting both in studying how predictions can be obtained how do algorithms perform in practice.
- In real time scheduling theory with preemption usually means preemption with resumming. What result can be obtained with restarting or revoking? What results can be improved with restarting or revoking noting that the offline benchmark can also use resumming whereas with revoking or restaring the benchmark does not need to preempt.

Comments on the magical *RobustMatch* algorithm

Raghvendra's deterministic *RobustMatch* algorithm (Algorithm 7.7.6 in the text) is based on an alternating path analysis inspired by the *Permutatation* algorithm. As we stated, for arbitrary metrics the algorithm obtains the optimum ratio for the online case (i.e.,adversarial order) and obtains $O(\log k)$ in the random order model.
The algorithm will maintain an online solution M and an approximate offline solution M^* that it is created incrementally. The algorithm critically uses the following concept:

Definition

Let P be an augmenting path in a matching M (between servers and requests). The t-net cost of P is defined as
$\phi_t(P) = t\left(\sum_{(u.v)\in P\setminus M} d(u.v)\right) - \sum_{(u,v)\in P\cap M} d(u,v)$.

Continuing the comments on the deterministic *RobustMatch* algorithm for atrbitrary metrics.

Note that when $t = 1$ the t-net cost is the cost of a min cost augmenting path P to update a matching. We will now use a path P of minimum t-net cost to guide the updating of the online algorithm. We view paths and matchings as sets of edges. While the statement of the algorithm is reasonably natural, the analysis in each of its applications is different and quite technical.

As t increases the cost to the algorithm decreases but the offline approximation gets worse. One might expect that there should be some best choice of t to balance the online cost and the offline approximation. But surprisingly, the competitve ratio improves with larger values of t. Namely, Raghvendra [2016] proves the following competitive ratios:

- For online algorithms (i.e., adversarial order) the ratio is $(2 + \frac{2}{t-1}) \cdot n - (1 - \frac{2}{t-1})$
- For random order algorithms, the ratio is $(2 + \frac{2}{t-1}) \cdot H_n - (1 - \frac{2}{t-1})$

The *RobustAlgorithm*

Algorithm 7.7.6 Algorithm *RobustMatching* parameterized by t for the Min Cost Perfect Matching problem.

procedure *RobustMatching*(t)
- $A \leftarrow S$ ▷ the set of available servers
- $M_0 \leftarrow \emptyset$ ▷ the matching constructed so far
- $M^* \leftarrow \emptyset$ ▷ the optimal matching so far
- $i \leftarrow 1$
- **while** $i \leq k$ **do**
  - **if** $\exists s_j \in A : s_j = r_i$ **then**
    - $M_i \leftarrow M_{i-1} \cup \{(r_i, s_j)\}$
    - $M^* \leftarrow M^* \cup \{(r_i, s_j)\}$
    - $A \leftarrow A \setminus \{s_j\}$
  - **else**
    - Let P be a t-net cost augmenting path in M^* of minimum cost starting at the current request r_i and ending at a server s_j in A.
    - $M_i \leftarrow M_{i-1} \cup \{(r_i, s_j)\}$
    - $M^* \leftarrow M^* \oplus P$
    - $A \leftarrow A \setminus \{s_j\}$.
  - $i \leftarrow i + 1$

The algorithm and its analysis for arbitrary metrics is in the [2016] paper by Raghvendra. The analysis for the line metric is in his [2018] paper.

Online input models for graph problems

For graph problems, there are two common ways to represent the input items. In one representation, edges $e = (u, v)$ are the input items. For weighted problems, the representation would give the weight of the edge and/or the weight of the adjacent vertices.

The other input representation is when vertices are the input items. Here the most common model is that an online vertex is given by the list of its adjacent vertices that have already arrived. We denote this input model as VAM-PH. The less common model is when the entire vertex adjacency list is given, denoted VAM-FI.

Note: When studying the problems for biprtite graphs, we so far have used the one-sided model where the offline nodes are known in advance.

For vertex cover, we will use the edge model.
For graph coloring we will use the VAM-PH input representation.

Unweighted and weighted vertex cover

So far, we have discussed two graph theoretic problems where we have online algorithms with optimal or good competitive ratios; namely, unweighted maximum bipartite matching and min cost metric matching. Both have optimal offline algorithms.

One other graph problem for which we have a good randomized and deterministic competitive ratios is the *NP*-hard vertex cover problem. (See Section 7.8.) Here we are assuming edges (u_i, v_i) are arriving online. For the unweighted case, there is a simple deterministic 2-competitive algorithm (i.e. take both vertices if the edge is not yet covered). This is the optimal deterministic ratio.

For the weighted case (where vertices have weights), there is a randomized algorithm (which I would call the natural randomized greedy algorithm with respect to the weights of the adjacent vertices) that obtains a 2-competitive ratio. That is, cover an uncovered edge $e = (u, v)$ by u with probability $\frac{w(v_i)}{w(u_i)+w(v_i)}$ and otherwise cover with v. This algorithm is due to Ausiello et al in their textbook "Complexity and Approximations".

A deterministic algorithmm for weighted vertex cover

The obvious deterministic greedy algorithm covers an uncovered edge (u_i, v_i) by u_i if $w(u_i) \leq w(v_i)$, and otherwise covers with v_i. It is with respect to this deterministic algorithm that (in hindsight) we defined the previous "natural randomized elgorithm".

But this determinsitc algorithm is not competitive having competitive ratio $\Omega(n)$. In particular, consider the execution of this algorithm on a weighted star graph where the weight of each leaf is slightly larger than the weight of the center node.

In an analogy to the ski rental problem and the BreakEven algorithm, for the star graph we can think of our choice to cover (u, v_i) with a leaf v_i as a rental cost and think of using the center as a buying cost. Following the analogy, we want to keep track of the rental costs for using the v_i and buy the center node u when the accumulated weights of choosing the $\{v_i\}$ "breaks even" with the weight of u.

A deterministic vertex cover algorithm continued

We can implement this "break even" idea on an arbitrary graph by maintaining dynamic weights $\widetilde{w}(v)$. Now to cover an edge (u, v), we choose the vertex (say v) with the smaller dynamic weight and then update the dynamic weight of u by subtracting the dynamic weight of v from the dynamic weight of u.

The algorithm is proven to be 2-competitive in our text Theorem 7.3.

A similar use of updated weights for the weighted vertex cover problem is given in Clarkson [1983] where he gives a 2-approximation greedy (i.e adaptive priority) algorithm where now vertices are the input items. I am not sure if the weighted vertex cover problem has been studied in the online model. Howverer, in the VAM-PH model (where algorithm the algorithm must either accept or reject each online vertex without knowledge of future vertices), it is not difficult to see that there cannot be a constant competitive algorithm.

The 2-competitive deterministic algorithm for weighted vertex cover

Algorithm 7.3.3 Augmented greedy algorithm for the online Weighted Vertex Cover problem.

procedure *AugmentedGreedyVC*
  $S \leftarrow \emptyset$ ▷ S will be the vertices in the vertex cover
  $F \leftarrow \emptyset$ ▷ an auxiliary data structure to help with the proof
  $i \leftarrow 1$
  while $i \leq m$ **do**
    New online edge $e_i = \{u_i, v_i\}$ arrives according to $\prec$ together with vertex weights $w(u_i)$ and $w(v_i)$
    if $S \cap \{u_i, v_i\} = \emptyset$ **then** ▷ if vertices adjacent to e_i are not in S
      if u_i appears for the first time **then**
        $\widetilde{w}(u_i) \leftarrow w(u_i)$
      if v_i appears for the first time **then**
        $\widetilde{w}(v_i) \leftarrow w(v_i)$
      if $\widetilde{w}(u_i) \leq \widetilde{w}(v_i)$ **then**
        $S \leftarrow S \cup \{u_i\}$
        $\widetilde{w}(v_i) \leftarrow \widetilde{w}(v_i) - \widetilde{w}(u_i)$
        add directed edge (u_i, v_i) to F
      else
        $S \leftarrow S \cup \{v_i\}$
        $\widetilde{w}(u_i) \leftarrow \widetilde{w}(u_i) - \widetilde{w}(v_i)$
        add directed edge (v_i, u_i) to F
          ▷ we greedily selected smaller vertex according to augmented weights
    $i \leftarrow i + 1$

Online colouring of graphs

Many graph theoretic problems are *NP*-hard for arbitrary graphs and indeed many are *NP*-hard to obtain an approximation significantly better than a naive algorithm. Although that doesn't rule out online and priority algorithms with good competitive or priority ratios, we don't have much evidence of such algorithms. Potential project?

One *NP* hard to approximate is vertex colouring where it is known to be *NP* hard to obtain an approximation $n^{1-\epsilon}$ for any $\epsilon > 0$.
Corresponding to the NP-hardness of coloring , Halldorsson and Szegedy [1994] [prove the following negative result for online colouring of arbitrary graphs.

Theorem: No deterministic or randomized online algorithm can obtain a competitive ratio better than $\frac{2n}{\log^2 n}$ where n is the number of vertices.
Before I sketch the proof, I want to point out that graph colouring can be online reduced to vector bin packing as shown in section 8.2. This immediately gives a lower bound for online vector bin packing.

The reduction of colouring to vector bin packing

There is an online reduction of graph coloring to vector bin packing. Given a graph $G = (V, E)$ with $n = |V|$ vertices, we construct a $d = n$ dimensional vector bin packing instance as follows: For each vertex v_i, we create a vector $\mathbf{v}'_i$ such that $v'_i(i) = 1$ and $v'_i(j) = 1/n$ if $(v_i, v_j) \in E$ and $j < i$ and $v'(j) = 0$ otherwise. Here $v'(j)$ is the j^{th} coordinate of the vector $\mathbf{v}'$.
Then for any $S \subseteq [1, n]$, $v'(S) = \{\mathbf{v}'_i : i \in S\}$ can be packed in a single bin if and only if $V_S = \{v_i : i \in S\}$ is an independent set in G (i.e., can be colored with one color).

Theorem: For any function $q(d)$ there exists a constant c such that for any randomized algorithm ALG, there exist instances I where $\mathbb{E}[ALG(I)] \geq c\frac{d}{\log^3 d} \cdot OPT + q(d)$. The lower bound assumes that $n = |V|$ is known in advance. (There is a somewhat better but more technically involved $\Omega(\frac{d}{\log^2 d})$ asymptotic lower bound .)
Aside: What problems *online reduce* to what other problems. Are there classes of online problems for which there are "complete" problems?

The proof of the HS lower bound

We start with the proof for deterministic algorithms which can then be extended to randomized algorithms.

We view the construction of the nemesis graph as a game betwen the Algorithm and an Adaptive Online Adversary. Namely, the adversary will present each online vertex v_i by giving its pre-neighbourhood N^{-1}. The algorithm will provide a colour $b(v_i)$ and then the adversary will announce its colour $c(v_i)$. In the more standard game, the adversary need not determine its solution untile the end. This is, of course, what we called an adaptive online adversary wrt randomized algorithms.

We think of placing the online vertices in bins. Let $H(j) = \{c(v) : b(v) = j\}$. These are the colours used by the adversary to colour nodes with online colour $b(v)$. As time goes on, bin j will be eliminated when $|H(j)| = n/2$.

Then let $S = \binom{[k]}{k/2} \setminus \{H(j)\}$ for some $j < i$. The adversary then chooses $N^{-1}(v_i) = \{v_j : j < i \text{ and } c(v_j) \notin S\}$; the adversary chooses a colour in S.

Continuation of the HS lower bound

The game continues as long as there is some bin that is still available. Thus, the game goes on for at least $n = \frac{k}{2}\binom{[k]}{k/2}$ iterations.

This means that the algorithm has used $\binom{[k]}{k/2}$ colours while the adversary has used at most k colours. This implies we can set $k \approx \log n$ resulting in the algorithm using $\frac{2n}{\log n}$ colours which establishes the competitive ratio.

The argument can be extended using the Yao Principle. The adverssary no longer can deterministically choose S since it doesn't know what are the available colours. Instead, an oblivious adverary chooses a random set S of colours of size $k/2$ and a random colour in S. It can be shown that for $n \leq 2^{k/4}$ iterations. the probability that the adversary choose a non-admissable colour is sufficiently small.

Online coloring of graph classes

Given that many graph probelms are hard to approximate, one then often studies classes of graphs. In this regard there are many results concerning online vertex colouring of different graph classes.

In the text we discuss trees, bipartite graphs, d-inductive graphs and interval graphs. We have the following defintions:

- A d-inductive graph is such that there is an ordering of the vertices $v_1, v_2, \ldots, v_n$ such that v_i has degree at most d when restricted to $Nbhd(v_i) \cap \{v_{i+1} \ldots, v_n\}$.
- A chordal graph has many equivalent definitions. See the lecture by Don Knuth (https://www.youtube.com/watch?v=txaGsawIjjA) on trees and chordal graphs.
  One definition that is most relevent for us is the following: A chordal graph has a *perfect elimination ordering*; namely, there exist an ordering $v_1, \ldots, v_n$ such that the induced subgraph of $Nbhd(v_i) \cap \{v_{i+1} \ldots, v_n\}$ is a clique.
  Knuth: Chordal graphs are "probably the next most important class (after trees) of graphs".

Examples of d-inductive and chordal graphs

- Trees are 1-inductive and trees are also bipartite.
- Planar graphs are 5-inductive.
- Chordal graphs are $\chi(G)$ inductive where $\chi(G)$ is the colouring number of the graph.
- Trees and interval graphs are chordal graphs. An interval graph is the intersection graph induced by intervals on the line. The perfect elimination ordering for intervals is to sort intervals by non-decreasing finishing times.

Possible project: Whenever we discuss a problem for a graph class, we can consider online, ROM, or priority algorithms. When there is an online algorithm, is there a better (wrt approximation) priority algorithm? When there is a priority algorithm, is there also an online algorithm with a competitive ratio as good or nearly as good as the priority ratio. And when can a result for a graph class be extended to a larger class?

I wasn't aware of ROM or recourse results for colouring. So I did a search and immediately found one paper for colouring trees by Frei et al [2024].

Coloring trees and Bipartite Graphs

For coloring trees we have the following (almost) optimal online result:

Theorem: Every tree can be online colored with $\lfloor \log n + 1 \rfloor$ colors by the *FirstFit* algorithm.

This then shows that the competitve ratio is $\frac{\lfloor \log n+1 \rfloor}{2}$ since trees are bipartite graphs and bipartite graphs are precisely the graphs that are 2-colourable. This is (almost exactly) a tight bound since it is shown that:

For every deterministic online colouring algorithm, there is a a graph that requires $\frac{\lfloor \log n \rfloor}{2}$ colours.
The lower bound follows from the following inductive statement that holds for any deterministic colouring algorithm ALG. :
For every deterministic algorith and for all $k \geq 1$, the adversary can create disjoint trees $T_1, T_2, \ldots$ wuch that the akgorithm has used k colours to colour the roots of these trees T_i and the total number of nodes in all these trees is at most $2^k - 1$.

The lower bound proof for online tree colouring.

The base of the induction is $k = 1$ in which we have a single node tree. For the induction step, assume true for k and extend to $k + 1$. Let $T_1, T_2 \ldots, T_m$ be the trees whose roots are coloried with k different colors. Call thess colors S. Then after all the nodes corresponding to the trees $\{T_i\}$ have been colored, create a second list of trees $T_1', T_2', \ldots$. Then one of two cases can occur:

1. One of the T_i' trees is colored with a color not in S. Then we are done since the algorithm has used $k + 1$ colors and the combined size is at most i $2(2^k - 1) < 2^{k+1} - 1$.
2. The roots $r_1, r_2, \ldots$ of the T_i' trees are colored by the k colors in S. Then the adversary creates a new vertex connected to each of the r_i so that the algorithm must use a new colour. The total number of nodes in all the t_i and T_i' trees (including the new root) is at most $2(2^k - 1) + 1 = 2^{k+1} - 1$.

   The lower bound on the competitive ratio follows since trees can be coloured with 2 colours and the algorithm has used $k = \log(n + 1)$ colours.