CSC2420 Spring 2015: Lecture 10

Allan Borodin

March 19, 2015

Announcements and todays agenda

- Announcements
  - The first 4 questions of the final assignment are now posted. I plan to add at most one or two more questions.
- Todays agenda
  1. Filing in some details for the randomized rounding of LP.
  2. Review rounding of vector program for Max-2-Sat.
  3. Review of Random walks and application to 2-SAT.
  4. Extending the random walk algorithm to k-SAT.
  5. Whirlwind tour of online maximum matching and adwords problems.
  6. Sublinear time algorithms

Filing in some details of the IP/LP for Max-Sat

- Recall the weighted Max-Sat problem formulated as a $\{0,1\}$ IP.
- Let F be a CNF formula with n variables $\{x_i\}$ and m clauses $\{C_j\}$. The Max-Sat formulation is :
  maximize $\sum_j w_j z_j$
  subject to $\sum_{\{x_i \text{ is in } C_j\}} y_i + \sum_{\{\bar{x}_i \text{ is in } C_j\}} (1 - y_i) \geq z_j$
  $y_i \in \{0,1\}$; $z_j \in \{0,1\}$
- The y_i variables correspond to the propositional variables and the z_j correspond to clauses.
- The relaxation to an LP is $y_i \geq 0$; $z_j \in [0,1]$.

Randomized rounding of the y_i variables

- Let $\{y_i^*\}, \{z_j^*\}$ be the optimal LP solution,
- Set $\tilde{y}_i = 1$ with probability y_i^*.

Theorem

Let C_j be a clause with k literals and let $b_k = 1 - (1 - \frac{1}{k})^k$. Then $Prob[C_j$ is satisifed $]$ is at least $b_k z_j^*$.

- The theorem shows that the contribution of the j^{th} clause C_j to the expected value of the rounded solution is at least $b_k w_j z_j^*$ compared to the LP-OPT contribution of $w_j z_j^*$.
- Note that b_k is a decreasing function (of k) and converges to (and is always greater than) $1 - \frac{1}{e}$ as k increases. It follows that the expected value of the rounded solution is at least $(1 - \frac{1}{e})$ LP-OPT $\approx .632$ LP-OPT.
- Taking the max of this IP/LP and the naive randomized algorithm results in a $\frac{3}{4}$ approximation algorithm that can be derandomized.

Proof sketch of previous theorem

- We consider an arbitrary clause C_j with k literals. By replacing any $\bar{x}_i$ by x_i we can assume the clause has no negated variable and by renaming we can assume that $C_j = x_1 \vee x_2 \ldots \vee x_k$.
- C_j is satisfed if $x_1, \ldots x_k$ are not all set *false*. By the rounding procedure, the probability of this happening is

$$1 - \prod_{i}^{k}(1 - y_i) \geq \left(\frac{1 - \sum_i^k (1 - y_i)}{k} \right)^k$$

by the arithmetic-geometric mean inequality

$$\geq 1 - (1 - \frac{z_j^*}{k})^k$$

by the constraint $\sum_i^k y_i \geq z_j$ and the concavity of the function $g(z) = 1 - (1 - \frac{z}{k})^k$.

Review:The quadratic program for Max-2-Sat

- We introduce {-1,1} variables y_i corresponding to the propositional variables. We also introduce a homogenizing variable y_0 which will correspond to a constant truth value. That is, when $y_i = y_0$, the intended meaning is that x_i is set *true* and *false* otherwise.
- We want to express the $\{-1, 1\}$ truth value $val(C)$ of each clause C in terms of these $\{-1, 1\}$ variables.

1. $val(x_i) = (1 + y_i y_0)/2$
   $val(\bar{x}_i) = (1 - y_i y_0)/2$
2. If $C = (x_i \vee x_j)$, then $val(C) = 1 - val(\bar{x}_i \wedge \bar{x}_j) = 1 - (\frac{1-y_iy_0}{2})(\frac{1-y_jy_0}{2}) = (3 + y_iy_0 + y_jy_0 - y_iy_j)/4 = \frac{1+y_0y_i}{4} + \frac{1+y_0y_j}{4} + \frac{1-y_iy_j}{4}$
3. If $C = (\bar{x}_i \vee x_j)$ then $val(C) = (3 - y_iy_0 + y_jy_0 + y_iy_j)/4$
4. If $C = (\bar{x}_i \vee \bar{x}_j)$ then $val(C) = (3 - y_iy_0 - y_jy_0 - y_iy_j)/4$

The quadratic program for Max-2-Sat continued

- The Max-2-Sat problem is then to maximize $\sum w_k val(C_k)$ subject to $(y_i)^2 = 1$ for all i
- By collecting terms of the form $(1 + y_i y_j)$ and $(1 - y_i y_j)$ the max-2-sat objective can be represented as the strict quadratic objective: $\max \sum_{0 \le i < j \le n} a_{ij}(1 + y_i y_j) + \sum b_{ij}(1 - y_i y_j)$ for some appropriate a_{ij}, b_{ij}.
- Like an IP this integer quadratic program cannot be solved efficiently.

The vector program relaxation for Max-2-Sat

- We now relax the quadratic program to a vector program where each y_i is now a unit length vector $\mathbf{v}_i$ in $\Re^{n+1}$ and scalar multiplication is replaced by vector dot product. This vector program can be (approximately) efficiently solved (i.e. in polynomial time).
- The randomized rounding (from $\mathbf{v}_i^*$ to y_i) proceeds by choosing a random hyperplane in $\Re^{n+1}$ and then setting $y_i = 1$ iff $\mathbf{v}_i^*$ is on the same side of the hyperplane as $\mathbf{v}_0^*$. That is, if $\mathbf{r}$ is a uniformly random vector in $\Re^{n+1}$, then set $y_i = 1$ iff $\mathbf{r} \cdot \mathbf{v}_i^* \geq 0$.
- Let $\alpha = \frac{2}{\pi} \min_{\{0 \leq \theta \leq \pi\}} \frac{\theta}{(1 - cos(\theta))} \approx .87856$. It follows that $\frac{\theta}{\pi} \geq \alpha(\frac{1 - cos\theta}{2})$
- The rounded solution then has expected value $2 \sum a_{ij} Prob[y_i = y_j] + b_{ij} Prob[y_i \neq y_j]$; $Prob[y_i \neq y_j] = \frac{\theta_{ij}}{\pi}$ where θ_{ij} is the angle between $\mathbf{v}_i^*$ and $\mathbf{v}_j^*$.

The approximation ratio (in expectation) of the rounded solution

$\mathbf{E}$[rounded solution] $\geq \alpha \cdot (OPT_{VP})$.

Random walks on graphs and 2-SAT walk

- Let $G = (V, E)$ be a connected, non-bipartite, undirected graph with $|V| = n$ and $|E| = m$. A uniform random walk induces a Markov chain M_G as follows: the states of M_G are the vertices of G; and for any $u, v \in V$, $P_{uv} = 1/deg(u)$ if $(u, v) \in E$, and $P_{uv} = 0$ otherwise.
- Denote by $(d_1, d_2, \ldots, d_n)$ the vertex degrees. M_G has a stationary distribution $(d_1/2m, \ldots, d_n/2m)$.
- Let $C_u(G)$ be the expected time to visit every vertex, starting from u and define $C(G) = \max_u C_u(G)$ to be the *cover time* of G.

Theorem: Aleliunas et al [1979]

Let G be a connected undirected graph. Then

1. For each edge (u, v), $C_{u,v} \leq 2m$,
2. $C(G) \leq 2m(n-1)$.

- It follows that the 2-SAT random walk has expected time at most $2n^2$. to find a satisfying assignment in a satisfiable formula. Can use Markov inequality to obtain probability of not finding satisfying assignment.

Extending the random walk idea to k-SAT

- The random walk 2-Sat algorithm might be viewed as a drunken walk (and not an algorithmic paradigm). We cn also view the approach as a local search algorithm that doesn't know when it is making progress on any iteration but does have confidence that such an exploration of the local neighborhood is likely to be successful over time.
- We want to extend the 2-Sat algorithm to k-SAT. However, we know that k-SAT is NP-complete for $k \geq 3$ so our goal now is to improve upon the naive running time of 2^n, for formulas with n variables.
- In 1999, following some earlier results, Schöning gave a very simple (a good thing) random walk algorithm for k-Sat that provides a substantial improvement in the running time (over the naive 2^n exhaustive search) and this is still almost the fastest (worst case) algorithm known.
- This algorithm was derandomized by Moser and Scheder [2011].
- Beyond the theoretical significance of the result, this is the basis for various Walk-Sat algorithms that are used in practice.

Schöning's k-SAT algorithm

The algorithm is similar to the 2-Sat algorithm with the difference being that one does not allow the random walk to go on too long before trying another random starting assignment. The result is a one-sided error alg running in time $\tilde{O}[(2(1-/1k)]^n$; i.e. $\tilde{O}(\frac{4}{3})^n$ for 3-SAT, etc.

Randomized k-SAT algorithm

```
Choose a random assignment $\tau$
Repeat 3n times     % n = number of variables
If $\tau$ satisfies $F$ then stop and accept
Else Else Let $C$ be an arbitrary unsatisfied clause
  Randomly pick and flip one of the literals in $C$
End If
```

Claim

If F is satisfiable then the above succeeds with probability p at least $[(1/2)(k/k-1)]^n$. It follows that if we repeat the above process for t trials, then the probability that we fail to find a satisfying assignment is at most $(1-p)^t < e^{-pt}$. Setting $t = c/p$, we obtain error probability $(\frac{1}{e})^c$.

Randomized online bipartite matching and the adwords problem

- We return to online algorithms and algorithms in the random order model (ROM). Here we have already seen evidence of the power of the ROM model (over all deterministic online algorithms) in the context of the secretary problem. (See lecture 2.)
- As we suggested in lecture 2, another nice sequence of results begins with a randomized online algorithm for bipartite matching due to Karp, Vazirani and Vazirani [1990].
- We very quickly overview some results in this area as it represents a topic of current interest.
- In the online bipartite matching problem, we have a bipartite graph G with nodes $U \cup V$. Nodes in U enter online revealing all their edges. A deterministic greedy matching produces a maximal matching and hence a $\frac{1}{2}$ approximation.

Bipartite matching and the secretary problem.

- There are two weighted versions of the problem:
  1. In the vertex weighted case, the vertices $v \in V$ each have a weight $w(v)$ and the goal is to then to maximize the weight of a matching where each edge (u, v) has weight $w(v)$ for each online vertex u.
  2. In more general edge weighted case, the edges of all online arriving nodes u are revelaed along with their weights $w(u, v)$ for each edge (u, v).
- Edge weighted online matching can be seen as an extension of the secretary problem where the offline vertex set is just one node. But in online matching, we are maximizing the *unweighted* number of edges or the weight of a matching (in the weighted cases) and not the probability of obtaining the best match.
- Any *deterministic online* (resp.*priority*) algorithm for unweighted matching cannot approximate better than $\frac{1}{2}$ (resp $\frac{1}{2} + \epsilon$) even for degree at most 2 (resp. degree $(n + 1)/2$)).
- Recall: For secretary problem (and hence edge weighted matching), there cannot be an *online* deterministic or randomized constant approximation algorithm.

The randomized ranking algorithm for unweighted bipartite matching

- The algorithm chooses a random permutation of the nodes in V and then when a node $u \in U$ appears, it matches u to the highest ranked unmatched $v \in V$ such that (u, v) is an edge (if such a v exists).
- The Ranking algorithm achieves an (expected) approximation of $1 - 1/e \approx .63$
- Aside: making a random choice for each u is still only a $\frac{1}{2}$ approx.
- The analysis of this algorithm can be used to show that there is a deterministic greedy algorithm in the ROM model.
- That is, let $\{v_1, \ldots, v_n\}$ be any fixed ordering of the vertices and let the nodes in U enter randomly, then match each u to the first unmatched $v \in V$ according to the fixed order.
- To argue this, consider fixed orderings of U and V; the claim is that the matching will be the same whether U or V is entering online.

The KVV result and recent progress

KVV Theorem

Ranking provides a $(1 - 1/e)$ approximation.

- Original analysis is not rigorous.
- There is an alternative proof (and extension) by Goel and Mehta [2008], and other proofs (e.g. in Birnbaum and Mathieu [2008]).
- KVV show that the $(1 - 1/e)$ bound is essentially tight for any randomized online (i.e. adversarial input) algorithm. In the ROM model, Goel and Mehta state inapproximation bounds of $\frac{3}{4}$ (for deterministic) and $\frac{5}{6}$ (for randomized) algorithms.
- In the ROM model, Karande, Mehta, Tripathi [2011] show that Ranking achieves approximation at least .653 (beating $1 - 1/e$) and no better than .727.

Some comments on the Birnbaum and Mathieu proof

- The worst case example a (n, n) graph with a perfect matching.
- In particular, for $n = 2$, the precise expected competitive (i.e. approximation) ratiois $\frac{3}{4}$. The inapproximation can be seen by using the Yao principle for obtaining bounds on randomized algorithms.

The main lemma in the analysis

Let x_t be the probability (over the random permutations of the vertices in V) that the vertex of rank t is matched. Then $1 - x_t \leq \frac{1}{n} \sum_{s=1}^{t} x_s$

- Letting $S_t = \sum_{s=1}^{t} x_s$ the lemma can be restated as $S_t(1 + 1/n) \geq 1 + S_{t-1}$ fo all t. Given that the graph has a perfect matching, the expected competitive ratio is S_n/n. It is shown that $\frac{1}{n} S_n \geq 1 - (1 - \frac{1}{n+1})^n \rightarrow 1 - 1/e$.

Getting past the $(1 - 1/e)$ bound

- As discussed in the secretary problem, the ROM model can be considered as an example of what is called stochastic optimization in the OR literature. There are other stochastic optimization models that are perhaps more natural, namely i.i.d sampling from known and unknown distributions.
- Feldman et al [2009] study the known distribution case and show a randomized algorithm that first computes an optimal offline solution (in terms of expectation) and uses that to guide an online allocation.
- They achieve a .67 approximation (improved to .699 by Bahmani and Kapralov [2010] and to .705 by Manshadi et al [2011] and also show that no online (randomized) algorithm can achieve better than $26/27 \approx .963$. Inapporximation has been improved to .823 by Manshadi et al. This implies the same hardness for the unknown distribution and ROM models.
- In the unknown distrtibution model, Manshadi et al achieve a .702 approximation(almost matching their known distribution result).

Extensions of online bipartite matching

- Weighted online matching
- Adwords
- Applying the ROM model to matching problems
- Applying the determinstic priority to matching problems
- Stochastic matching
- Online with Reassignments

The adwords problem: an extension of bipartite matching

- In the (single slot) adwords problem, the nodes in U are queries and the nodes in V are advertisers. For each query q and advertiser i, there is a bid $b_{q,i}$ representing the value of this query to the advertiser.
- Each advertiser also usually has a hard budget B_i which cannot be exceeded. The goal is to match the nodes in U to V so as to maximize the sum of the accepted bids without exceeding any budgets. Without budgets and when each advertiser will pay for at most one query, the problem then is edge weighted bipartite matching.
- In the online case, when a query arrives, all the relevant bids are revealed.

Some results for the adwords problem

- Here we are just considering the combinatorial problem and ignoring game theoretic aspects of the problem.
- The problem has been studied for the special (but well motivated case) that all bids are small relative to the budgets. As such this problem is incomparable to the matching problem where all bids are in $\{0,1\}$ and all budgets are 1.
- For this small bid case, Mehta et al [2005) provide a deterministic online algorithm achieving the $1 - 1/e$ bound and show that this is optimal for all randomized online algorithms (i.e. adversarial input).

Greedy for a class of adwords problems

- Goel and Mehta [2008] define a class of adwords problems which include the case of small budgets, bipartite matching and b-matching (i.e. when all budgets are equal to some b and all bids are equal to 1).
- For this class of problems, they show that a deterministic greedy algorithm achieves the familiar $1 - 1/e$ bound in the ROM model. Namely, the algorithm assigns each query (.e. node in U) to the advertiser who values it most (truncating bids to keep them within budget and consistently breaking ties). Recall that Ranking can be viewed as greedy (with consistent tie breaking) in the ROM model.

Vertex weighted bipartite matching

- Aggarwal et al [2011] consider the vertex weighted version of the online bipartite matching problem where as we stated, the vertices $v \in V$ all have a known weight w_v and the goal is now to maximize the weighted sum of matched vertices in V when again vertices in U arrive online.
- This problem can be shown to subsume the adwords problem when all bids $b_{q,i} = b_i$ from an advertiser are the same.
- It is easy to see that Ranking can be arbitrarily bad when there are arbitrary differences in the weight. Greedy (taking the maximum weight match) can be good in such cases. Can two such algorithms be somehow combined? Surprisingly, Aggarwal et al are able to achieve the same 1-1/e bound for this class of vertex weighted bipartite matching.

The vertex weighted online algorithm

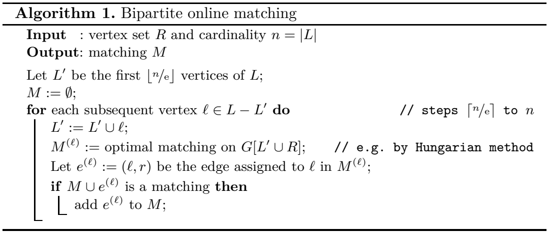
The perturbed greedy algorithm

For each $v \in V$, pick r_v randomly in $[0, 1]$
Let $f(x) = 1 - e^{1-x}$
When $u \in U$ arrives, match u to the unmatched v (if any) having the highest value of $w_v * f(r_v)$. Break ties consistently.

In the unweighted case when all w_v are identical this is the Ranking algorithm.

The edge weighted algorithm in the ROM model

Kesselheim et al [ESA 2013] show how to extend the ideas of the ROM secretary algorithm to obtain a $\frac{1}{e}$ approximation to the edge weighted biparitite matching problem in the ROM model as well as extending this idea to set packing (i.e. combinatorial auctions).



Algorithm 1. Bipartite online matching

```
Input  : vertex set $R$ and cardinality $n = |L|$
Output: matching $M$
Let $L'$ be the first $\lfloor n/e \rfloor$ vertices of $L$;
$M := \emptyset$;
for each subsequent vertex $\ell \in L - L'$ do                // steps $\lceil n/e \rceil$ to $n$
    $L' := L' \cup \ell$;
    $M^{(\ell)}$ := optimal matching on $G[L' \cup R]$;    // e.g. by Hungarian method
    Let $e^{(\ell)} := (\ell, r)$ be the edge assigned to $\ell$ in $M^{(\ell)}$;
    if $M \cup e^{(\ell)}$ is a matching then
        add $e^{(\ell)}$ to $M$;
```

[Kesselheim et al edge weighted biparitite matching algorithm]

Online algorithms allowing reassignments

There is a substantial history of results in scheduling allowing various forms of preemption. In the same spirit, we can allow online algorithms to undo previous decisions at some cost or in some limited way.
In particular,

1. Bar-Noy et al [2001] and independently Erlebach and Spieksma [2012] consider the weighted interval scheduling problem and the weighted JISP problem and show constant approximations when previously accepted intervals can be deleted and the requirement is that a feasible schedule must always be maintained.
2. This revocable acceptance model can be applied to any packing problem.
3. Gupta et al [2014] consider the makespan problem in the restricted machines problem and show that when each job has size 1 (resp. arbitrary size), an assignment can be maintained that is within twice (resp. a factor $O(\log\log mn)$) of the optimal makespan while using amortized $O(1)$ reassignments per job.
4. This doesn't say anything about the maximum matching problem as to an algorithm that could tradeoff some reassignments for an

Sublinear time and sublinear space algorithms

We continue to consider contexts in which randomization is provably necessary. In particular, we will study sublinear time algorithms and then the (small space) streaming model.

- An algorithm is sublinear time if its running time is $o(n)$, where n is the length of the input. As such an algorithm must provide an answer without reading the entire input.
- Thus to achieve non-trivial tasks, we almost always have to use randomness in sublinear time algorithms to sample parts of the inputs.
- The subject of sublinear time algorithms is a big topic and we will only present a very small selection of hopefully representative results.
- The general flavour of results will be a tradeoff between the accuracy of the solution and the time bound.
- This topic will take us beyond search and optimization problems.

A deterministic exception: estimating the diameter in a finite metric space

- We first conisder an exception of a “sublinear time” algorithm that does not use randomization. (Comment: “sublinear in a weak sense”.)
- Suppose we are given a finite metric space M (with say n points x_i) where the input is given as n^2 distance values $d(x_i, x_j)$. The problem is to compute the diameter D of the metric space, that is, the maximum distance between any two points.
- For this maximum diameter problem, there is a simple $O(n)$ time (and hence sublinear in n^2, the number of distances) algorithm; namely, choose an arbitrary point $x \in M$ and compute $D = \max_j d(x, x_j)$. By the triangle inequality, D is a 2-approximation of the diameter.
- I say sublinear time in a weak sense because in an explicitly presented space (such as d dimensional Euclidean space), the points could be explicitly given as inputs and then the input size is n and not n^2.

Sampling the inputs: some examples

- The goal in this area is to minimize execution time while still being able to produce a reasonable answer with sufficiently high probability.
- We will consider the following examples:
  1. Finding an element in an (anchored) sorted linked list [Chazelle,Liu,Magen]
  2. Estimating the average degree in a graph [Feige 2006]
  3. Estimating the size of some maximal (and maximum) matching [Nguyen and Onak 2008] in bounded degree graphs.
  4. Examples of property testing, a major topic within the area of sublinear time algorithms. See Dana Ron's DBLP for many results and surveys.

Finding an element in an (anchored) sorted list

- Suppose we have an array $A[i]$ for $1 \leq i \leq n$ where each $A[i]$ is a pair (x_i, p_i) with $x_1 = \min\{x_i\}$ and p_i being a pointer to the next smallest value in the linked list.
- That is, $x_{p_i} = \min\{x_j | x_j > x_i\}$. (For simplicity we are assuming all x_j are distinct.)
- We would like to determine if a given value x occurs in the linked list and if so, output the index j such that $x = x_j$.

A $\sqrt{n}$ algorithm for searching in an anchored sorted linked list

Let $R = \{j_i\}$ be $\sqrt{n}$ randomly chosen indices plus the index 1.
Access these $\{A[j_i]\}$ to determine k such that x_k is the largest of the accessed array elements less than or equal to x.
Search forward $2\sqrt{n}$ steps in the linked list to see if and where x exists

Claim:

This is a one-sided error algorithm that (when $x \in \{A[i]\}$) will fail to return j such that $x = A[j]$ with probability at most $1/2$.

Estimating average degree in a graph

- Given a graph $G = (V, E)$ with $|V| = n$, we want to estimate the average degree d of the vertices.
- We want to construct an algorithm that approximates the average degree within a factor less than $(2 + \epsilon)$ with probability at least $3/4$ in time $O(\frac{\sqrt{n}}{poly(\epsilon)})$. We will assume that we can access the degree d_i of any vertex v_i in one step.
- Like a number of results in this area, the algorithm is simple but the analysis requires some care.

The Feige algorithm

Sample $8/\epsilon$ random subsets S_i of V each of size (say) $\frac{\sqrt{n}}{\epsilon^3}$
Compute the average degree a_i of nodes in each S_i.
The output is the minimum of these $\{a_i\}$.

The analysis of the approximation

Since we are sampling subsets to estimate the average degree, we might have estimates that are too low or too high. But we will show that with high probability these estimates will not be bad. More precisely, we need:

1. Lemma 1: $Prob[a_i < \frac{1}{2}(1-\epsilon)\bar{d}] \leq \frac{\epsilon}{64}$
2. Lemma 2: $Prob[a_i > (1+\epsilon)\bar{d}] \leq 1 - \frac{\epsilon}{2}$

The probability bound in Lemma 2 is amplified as usual by repeated trials. For Lemma 1, we fall outside the desired bound if any of the repeated trials gives a very small estimate of the average degree but by the union bound this is no worse than the sum of the probabilities for each trial.

Understanding the input query model

- As we initially noted, sublinear time algorithms almost invariably sample (i.e. query) the input in some way. The nature of these queries will clearly influence what kinds of results can be obtained.
- Feige's algorithm for estimating the average degree uses only "degree queries"; that is, "what is the degree of a vertex v".
- Feige shows that in this degree query model, that any algorithm that acheives a $(2-\epsilon)$ approximation (for any $\epsilon > 0$) requires time $\Omega(n)$.
- In contrast, Goldreich and Ron [2008] consider the same average degree problem in the "neighbour query" model; that is, upon a query (v, j), the query oracle returns the j^{th} neighbour of v or a special symbol indicating that v has degree less than j. A degree query can be simulated by $\log n$ neighbour queries.
- Goldreich and Ron show that in the neighbour query model, that the average degree $\bar{d}$ can be $(1+\epsilon)$ approximated (with one sided error probability $2/3$) in time $O(\sqrt{n} poly(\log n, \frac{1}{\epsilon}))$
- They show that $\Omega(\sqrt{(}n/\epsilon))$ queries is necessary to achieve a $(1+\epsilon)$ approximation.

Approximating the size of a maximum matching in a bounded degree graph

- We recall that the size of any *maximal* matching is within a factor of 2 of the size of a maximum matching. Let m be smallest possible maximal matching.
- Our goal is to compute with high probability a *maximal* matching in time depending only on the maximium degree D.

Nguyen and Onak Algorithm

```
Choose a random permutation $p$ of the edges $\{e_j\}$
% Note: this will be done “on the fly” as needed
The permutation determines a maximal matching $M$ as given by the
   greedy algorithm that adds an edge whenever possible.
Choose $r = O(D/\epsilon^2)$ nodes $\{v_i\}$ at random
Using an “oracle” let $X_i$ be the indicator random variable for whether
   or not vertex $v_i$ is in the maximal matching.
Output $\tilde{m} = \sum_{i=1...r} X_i$
```

Performance and time for the maximal matching

Claims

1. $m \leq \tilde{m} \leq m + \epsilon$ n where $m = |M|$.
2. The algorithm runs in time $2^{O(D)}/\epsilon^2$

- This immediately gives an approximation of the *maximum* matching m^* such that $m^* \leq \tilde{m} \leq 2m^* + \epsilon n$
- A more involved algorithm by Nguyen and Onak yields the following result:

Nguyen and Onak maximum matching result

Let $\delta, \epsilon > 0$ and let $k = \lceil 1/\delta \rceil$. There is a randomized one sided algorithm (with probability $2/3$) running in time $\frac{2^{O(D^k)}}{\epsilon^{2^{k+1}}}$ that outputs a maximium matching estimate $\tilde{m}$ such that $m^* \leq \tilde{m} \leq (1+\delta)m^* + \epsilon n$.

Property Testing

- Perhaps the most prevalent and useful aspect of sublinear time algorithms is for the concept of property testing. This is its own area of research with many results.
- Here is the concept: Given an object G (e.g. a function, a graph), test whether or not G has some property P (e.g. G is bipartite) or is in some sense far away from that property.
- The tester determines with sufficiently high probability (say 2/3) if G has the property or is "ϵ-far" from having the property. The tester can answer either way if G does not have the property but is "ϵ-close" to having the property.
- We will usually have a 1-sided error in that we will always answer YES if G has the property.
- We will see what it means to be "ϵ-far" (or close) from a property by some examples.

Tester for linearity of a function

- Let $f : Z_n -> Z_n$; f is linear if $\forall x, y\ f(x+y) = f(x) + f(y)$.
- Note: this will really be a test for group homomorphism
- f is said to be ϵ-close to linear if its values can be changed in at most a fraction ϵ of the function domain arguments (i.e. at most ϵn elements of Z_n) so as to make it a linear function. Otherwise f is said to be ϵ-far from linear.

The tester

```
Repeat $4/\epsilon$ times
Choose $x, y \in Z_n$ at random
  If $f(x) + f(y) \neq f(x+y)$
  then Output $f$ is not linear
End Repeat If all these $4/\epsilon$ tests succeed then Output linear
```

- Clearly if f is linear, the tester says linear.
- If f is ϵ-far from being linear then the probability of detecting this is at least $2/3$.

Testing a list for monotonicity

- Given a list $A[i] = x_i, i = 1 \dots n$ of distinct elements, determine if A is a monotone list (i.e. $i < j \Rightarrow A[i] < A[j]$) or is ϵ-far from being monotone in the sense that more than $\epsilon * n$ list values need to be changed in order for A to be monotone.
- The algorithm randomly chooses $2/\epsilon$ random indices i and performs binary search on x_i to determine if x_i in the list. The algorithm reports that the list is monotone if and only if all binary searches succeed.
- Clearly the time bound is $O(\log n/\epsilon)$ and clearly if A is monotone then the tester reports monotone.
- If A is ϵ-far from monotone, then the probability that a random binary search will succeed is at most $(1 - \epsilon)$ and hence the probability of the algorithm failing to detect non-monotonicity is at most $(1 - \epsilon)^{\frac{2}{\epsilon}} \leq \frac{1}{e^2}$

Graph Property testing

- Graph property testing is an area by itself. There are several models for testing graph properties.
- Let $G = (V, E)$ with $n = |V|$ and $m = |E|$.
- Dense model: Graphs represented by adjacency matrix. Say that graph is ϵ-far from having a property P if more than ϵn^2 matrix entries have to be changed so that graph has property P.
- Sparse model, bounded degree model: Graphs represented by vertex adjacency lists. Graph is ϵ-far from property P is at least ϵm edges have to be changed.
- In general there are substantially different results for these two graph models.

The property of being bipartite

- In the dense model, there is a constant time one-sided error tester. The tester is (once again) conceptually what one might expect but the analysis is not at all immediate.

Goldreich, Goldwasser,Ron bipartite tester

Pick a random subset S of vertices of size $r = \Theta(\frac{\log(\frac{1}{\epsilon})}{\epsilon^2})$
Output bipartite iff the induced subgraph is bipartite

- Clearly if G is bipartite then the algorithm will always say that it is bipartite.
- The claim is that if G is ϵ-far from being bipartite then the algorithm will say that it is not bipartite with probability at least $2/3$.
- The algorithm runs in time quadratic in the size of the induced subgraph (i.e. the time needed to create the induced subgraph).

Testing bipartiteness in the bounded degree model

- Even for degree 3 graphs, $\Omega(\sqrt{n})$ queries are required to test for being bipartite or ϵ-far from being being bipartite. Goldreich and Ron [1997]
- There is a nearly matching algorithm that uses $O(\sqrt{n} poly(\log n/\epsilon))$ queries. The algorithm is based on random walks in a graph and utilizes the fact that a graph is bipartite iff it has no odd length cycles.

Goldreich and Ron [1999] bounded degree algorithm

```
Repeat $O(1/\epsilon)$ times
    Randomly select a vertex $s \in V$
    If algorithm $OddCycle(s)$ returns cylce found then REJECT
End Repeat
If case the algorithm did not already reject, then ACCEPT
```

- *OddCycle* performs $poly(\log n/\epsilon)$ random walks from s each of length $poly(\log n/\epsilon)$. If some vertex v is reached by both an even length and an odd length prefix of a walk then report cycle found; else report odd cycle not found