# CSC2422 Fall 2022: Algorithm Design, Analysis and Theory An introductory (i.e. foundational) level graduate course.

Allan Borodin

November 23, 2022; Lecture 9

## Brief Announcements and todays agenda

- Assignment 3 due Wednesday December 7, 1PM. So far I have posted two questions.

Todays agenda

- The SDP/vector program approach for Max-2-Sat
- Other applications of vector program rounding; cardinality constraints.
- A simple deterministic polynomial time algorithms for 2-SAT
- A relatively simple algorithm for 3-SAT that is more efficient than the naive alglorithm.
- The random walk algorithm for 2-SAT
- The random walk algorithm for *k*-Sat
- The exponential time hypothesis (ETH) and the strong exponential time hypothesis (SETH).

## The SDP/vector program approach: Max-2-Sat

- We briefly consider an important extension of the IP/LP approach, namely representing a problem as a strict quadratic program and then relaxing such a program to a vector program. Vector programs are known to be equivalent to semidefinite programs.
- For our purposes of just introducing the idea of this approach we will not discuss SDP concepts but rather just note that such programs (and hence vector programs) can be solved to arbitrary precision within polynomial time. This framework provides one of the most powerful optimization methods.
- We illustrate the approach in terms of the Max-2-Sat problem. A very similar algorithm and analysis produces the same approximation ratio for the Max-Cut problem.

I recommend the Vazirani text “Appoximation Algorithms” for this topic (and many other topics related to this course).

## 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 $\{0,1\}$ 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 $val(C) = 1 - val(\bar{x}_i \wedge \bar{x}_j) = 1 - (\frac{1-y_i y_0}{2})(\frac{1-y_j y_0}{2}) = (3 + y_i y_0 + y_j y_0 - y_i y_j)/4 = \frac{1+y_0 y_i}{4} + \frac{1+y_0 y_j}{4} + \frac{1-y_i y_j}{4}$
  3. If $C = (\bar{x}_i \vee x_j)$ then $val(C) = (3 - y_i y_0 + y_j y_0 + y_i y_j)/4$
  4. If $C = (\bar{x}_i \vee \bar{x}_j)$ then $val(C) = (3 - y_i y_0 - y_j y_0 - y_i y_j)/4$

## 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: $\max \sum_{0 \leq i < j \leq n} a_{ij}(1 + y_i y_j) + \sum b_{ij}(1 - y_i y_j)$ 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 $\mathbf{r} \cdot \mathbf{v}_i^* \geq 0$.
- The rounded solution then has expected value $2(\sum a_{ij} Prob[y_i = y_j] + \sum b_{ij} Prob[y_i \neq y_j])$ ; $Prob[y_i \neq y_j] = \frac{theta_{ij}}{\pi}$ where $\theta_{ij}$ is the angle between $\mathbf{v}_i^*$ and $\mathbf{v}_j^*$.

**The approximation ratio (in expectation) of the rounded solution**

Let $\alpha = \frac{2}{\pi} \min_{\{0 \leq \theta \leq \pi\}} \frac{\theta}{(1-cos(\theta))} \approx .87856$ and let $OPT_{VP}$ be the value obtained by an optimal vector program solution.
Then $\mathbf{E}[\text{rounded solution}] \geq \alpha \cdot (OPT_{VP})$.

## The densest subgraph problem

The densest subgraph problem is defined as follows:

Given a graph $G = (V, E)$, find a subset $V' \subseteq V$ so as to maximize $\frac{|e: u,v \in V'|}{|V'|}$; that is, to maximize the density (or equivalently the average degree) in a subgraph of $G$.

There is also a directed graph version of this problem. We will consider the undirected case.

## The densest subgraph problem

The densest subgraph problem is defined as follows:

Given a graph $G = (V, E)$, find a subset $V' \subseteq V$ so as to maximize $\frac{|e:u,v \in V'|}{|V'|}$; that is, to maximize the density (or equivalently the average degree) in a subgraph of $G$.

There is also a directed graph version of this problem. We will consider the undirected case.

The densest subgraph problems can be solved in polynomial time by a flow based algorithm as described in Lawler's 1976 text and improved in Gallo et al [1989]. There is also an LP duality based optimal method given in Charikar [2000] that is the starting point for the MapReduce $(1+\epsilon)$ approximation algorithm due to Bahmani, Goel and Munagala [2014].

The $(1+\epsilon)$ approximation follows a MapReduce $2(1+\epsilon)$ approximation by Bahmani, Kumar and Vassilvitskii [2012] based on the Charikar greedy 2-approximation. (Note that these papers use approximation ratios $\geq 1$.)

## Some comments on the densest subgraph problem

One immediate application is that the densest subgraph can be used to identify a “community” in the web or a social network. There are other applications as well in biological networks.

## Some comments on the densest subgraph problem

One immediate application is that the densest subgraph can be used to identify a "community" in the web or a social network. There are other applications as well in biological networks.

The $k$-densest subgraph problem asks for the densest subgraph $V'$ of size $|V'| = k$. As far as I know Feige, Peleg and Kortsarz [2001] have the current best approximation $O(n^{\frac{1}{3}+\delta})$ for the $k$-densest subgraph problem. It is also possible to achieve approximations slightly better than $n/k$ and in particular, for $k = \Theta(n)$.

## Some comments on the densest subgraph problem

One immediate application is that the densest subgraph can be used to identify a "community" in the web or a social network. There are other applications as well in biological networks.

The $k$-densest subgraph problem asks for the densest subgraph $V'$ of size $|V'| = k$. As far as I know Feige, Peleg and Kortsarz [2001] have the current best approximation $O(n^{\frac{1}{3}+\delta})$ for the $k$-densest subgraph problem. It is also possible to achieve approximations slightly better than $n/k$ and in particular, for $k = \Theta(n)$.

Obviously if one can (approximately) solve the $k$-densest subgraph problem then one can (approximately) solve the "densest subgraph with at least (or at most) $k$ vertices. But the converse is not apparentlly true. Andersen and Chellapilla [2009] show the following:

- There is a 3-approximation algorithm for the "at least $k$ vertices" variant.
- If there is a $\gamma$ approximatin for the "at most $k$ vertices" variant then there is an $\frac{\gamma^2}{8}$ approximation for the $k$-densest subgraph problem

## Set function maximization

Densest subgraph, max cut, max-di-cut and max-sat are problems where the goal is to maximize a non-monotone set function and hence (given that they are non-monotone) make sense in their *unconstrained* version.

Of course, similar to monotone set function maximization problems (such as max coverage), these problems also have natural constrained versions, the most studied version being a cardinality constraint.

More generally, there are other specific and arbitrary matroid constraints, other independence constraints, and knapsack constraints.

## Cardinality constrained set function maximization

Max-$k$-densest subgraph, max-$k$-cut, max-$k$-di-cut, max-$k$-uncut and max-$k$-vertex-coverage are the natural cardinality constrained versions of well studied graph maximization problems. They all are of the following form:

Given an edge weighted graph $G = (V, E, w)$ with non negative edge weightes $w : E \to \mathbb{R}$, find a subset $S \subseteq V$ with $|S| = k$ so as to maximize some set function $f(S)$. (Of course, In the unweighted versions, $w(e) = 1$ for all $e \in E$.)

For example, the objective in max-$k$-uncut is to find $S$ so as to maximize the edge weights of the subgraphs induced by $S$ and $V \setminus S$. That is, in a social network, divide the graph into two "communities".

**NOTE:** Max-$k$-sat is *not* the cardinality constrained version of max-sat in the same sense as the above problems. Although not studied (as far as I know), the analogous problem would be to find a set of propositional variables of cardinality $k$ so as to maximize the weights of the satisfied clauses.

## The SDP/vector program algorithms for the cardinality constrained problems

In what follows, I will briefly sketch some of the SDP based analysis in Feige and Langberg [2001]. This paper was proceeded and followed by a substantial number of important papers including the seminal Goemans and Williamson [1995] SDP approximation algorithm for max-cut.

(See , for example, Feige and Goemans [1995] and Frieze and Jerrun [1997] for proceeding work and Halperin and Zwick [2002], Han et al [2002] and Jäger and Srivastav for some improved and unifying results.)

There are also important LP based results such as the work by Ageev and Sviridenko [1999, 2004] that introduced *papage rounding*.
Many papers focus on the *bisection* versions where $k = n/2$ and also $k = \sigma n$ for some $0 < \sigma < 1$ for which much better approximations atre known relative to results for a general cardinality $k$ constraint.

## The Goemans and Williamson program algorithm for max-cut

As stated in Lecture 6, vector programs can be solved to arbitrary precision within polynomial time.

- We introduce {-1,1} variables $y_i$ corresponding to the vertex variables $x_i$. We also need a homogenizing variable $y_0$; the intended meaning is that vertex $v_i \in S$ and iff $y_i = y_0$.
- The max-cut problem can then be represented by tbhe following (strict) quadratic programming problem:

  Maximize $\frac{1}{2}\sum_{1\leq i<j\leq n} w_{ij}(1 - y_i y_j)$ subject to
  $y_i^2 = 1 \quad$ for $0 \leq i \leq n$
- This is relaxed to a vector program by introducing vectors on the unit sphere in $\mathbf{v}_i \in \mathbb{R}^{n+1}$ where now the scalar multiplication becomes the vector inner product.

# The rounding of the vector program

- 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 $\mathbf{r} \cdot \mathbf{v}_i^* \geq 0$.
- The rounded solution then has expected value $\sum_{1 \leq i < j \leq n} w_{ij} \mathbf{Pr}[\mathbf{v_i} \text{ and } \mathbf{v_j} \text{ are separated}] = \sum_{1 \leq i < j \leq n} w_{ij} \frac{\theta_{ij}}{\pi}$ where $\theta_{ij}$ is the angle between $\mathbf{v}_i^*$ and $\mathbf{v}_j^*$.

**The approximation ratio (in expectation) of the rounded solution**

Let $\alpha = \frac{2}{\pi} \min_{\{0 \leq \theta \leq \pi\}} \frac{\theta}{(1 - cos(\theta))} \approx .87856$ and let $OPT_{VP}$ be the value obtained by an optimal vector program solution.
Then $\mathbf{E}[\text{rounded solution}] \geq \alpha \cdot (OPT_{VP})$.

## Extending the vector program formulation for the cardinality constraint

The basic idea is to add an additional constraint:

$$\sum_{i=1}^{n} \mathbf{v}_i \mathbf{v}_0 = 2k - n$$

It turns out that it is sometimes important to define an improved relaxation by using instead (for all $j \in \{0, \ldots, n\}$) the "caraidnality constraints": $\sum_{i=1}^{n} \mathbf{v}_i \mathbf{v}_j = \mathbf{v_j} \mathbf{v}_0 (2k - n)$
For vectors $\mathbf{v}_j$ in the unit sphere, these constraints are equivalent to the constraints $\sum_{i=1}^{n} \mathbf{v}_i = \mathbf{v}_0 (2k - n)$

It also turns out that sometimes problems also use the following "triangle inequality constraints": $\mathbf{v}_i \mathbf{v}_j + \mathbf{v}_j \mathbf{v}_k + \mathbf{v}_k \mathbf{v}_i \geq -1$
and $\mathbf{v}_i \mathbf{v}_j + \mathbf{v}_j \mathbf{v}_k + \mathbf{v}_k \mathbf{v}_i \geq -1$ for all $i, j, k \in \{0, 1, \ldots, n\}$

## What else has to be done?

Skipping some important considerations (not used for the max-$k$-coverage and max-$k$-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

## What else has to be done?

Skipping some important considerations (not used for the max-$k$-coverage and max-$k$-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

The random hyperplane rounding insures the required probability that the round vectors will be separated. **BUT** this rounding does *not* enforce the desired $k$ cardinality constraint.

## What else has to be done?

Skipping some important considerations (not used for the max-$k$-coverage and max-$k$-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

The random hyperplane rounding insures the required probability that the round vectors will be separated. **BUT** this rounding does *not* enforce the desired $k$ cardinality constraint.

This is rectified by Feige and Langberg by modifying the s SDP results so as to penalize the resulting sets $S$ by a penalty depending on the deviation from the desired cardinality $k$.

## What else has to be done?

Skipping some important considerations (not used for the max-$k$-coverage and max-$k$-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

The random hyperplane rounding insures the required probability that the round vectors will be separated. **BUT** this rounding does *not* enforce the desired $k$ cardinality constraint.

This is rectified by Feige and Langberg by modifying the s SDP results so as to penalize the resulting sets $S$ by a penalty depending on the deviation from the desired cardinality $k$.

Namely, they run the SDP sufficiently many times and output the set that maximizes $Z = \frac{w(S)}{OPT_{SDP}} + \theta_1 \frac{n-|S|}{n-k} + \theta_2 \frac{|S|(2k-|S|)}{n^2}$ where $w(S)$ is the SDP rounded output, $OPT_{SDP}$ is the optimum relaxed value, and the $\theta$ are appropriately optimized scalars.

## The formulation for other set function maximization problems

The formulation and idea of the relaxation follows the same idea by mainly changing the objective function. (Recall the objective for max-2-sat.)

- For the max-$k$-densest subgraph problem, the objective (wrt. $y_i \in \{-1, 1\}$) is to maximize $\sum_{e_{ij} \in E} w_{ij} \frac{1+y_i y_0 + y_j y_0 + y_i y_j}{4}$
- The max-$k$-vertex-coverage problem a special case of the max coverage where each element (i.e. an edge) occurs in eactly two of the sets (i.e. vertices).

  The objective is to maximize $\sum_{e_{ij} \in E} w_{ij} \frac{3+y_i y_0 + y_j y_0 - y_i y_j}{4}$
  Here by monotocity we do not have to worry about outputs with $|S| < k$

  Now to compensate for $|S| > k$, we optimize $Z = \frac{w(S)}{OPT_{SDP}} + \theta \frac{n-|S|}{n-k}$.

## The results in Feige and Langberg

TABLE 1
Approximation ratios achieved on our four maximization problems. Our results appear in columns in which the SDP technique is mentioned.

| Problem | Technique | Approximation ratio | Range |
|---|---|---|---|
| Max-VC$_k$ | Random | $1-(1-k/n)^2$ | all $k$ |
| | Greedy | $\max(1-1/e, 1-(1-k/n)^2)$ | all $k$ |
| | LP | $\frac{3}{4}$ | all $k$ |
| | SDP | 0.8 | $k \geq n/2$ |
| | SDP | 0.8 | $k$ size of minimum VC |
| | SDP | $3/4 + \varepsilon$ | all $k$, universal $\varepsilon > 0$ |
| Max-DS$_k$ | Random | $\frac{k(k-1)}{n(n-1)}$ | all $k$ |
| | Greedy | $O(k/n)$ | all $k$ |
| | LP | $\frac{k}{n}(1-\varepsilon)$ | all $k$, every $\varepsilon > 0$ |
| | SDP | $k/n + \varepsilon_k$ | $k \sim n/2$ |
| Max-Cut$_k$ | Random | $\frac{2k(n-k)}{n(n-1)}$ | all $k$ |
| | LP | $\frac{1}{2}$ | all $k$ |
| | SDP | $1/2 + \varepsilon$ | all $k$, universal $\varepsilon > 0$ |
| Max-UC$_k$ | Random/LP | $1 - \frac{2k(n-k)}{n(n-1)}$ | all $k$ |
| | SDP | $1/2 + \varepsilon_k$ | $k \sim n/2$ |

## The results in Jäger and Srivastav

I think the following represents the latest improvements in cardinality constrained set function maximization for $k = \sigma n$ from Jäger and Srivastav [2005]

I think the following represents the latest improvements in cardinality constrained set function maximization for $k = \sigma n$ from Jäger and Srivastav [2005]

| Problem | $\sigma$ | Prev. | Our Method |
|---|---|---|---|
| MAX-$k$-CUT | 0.3 | 0.527 | 0.567 |
| MAX-$k$-UNCUT | 0.4 | 0.5258 | 0.5973 |
| MAX-$k$-DIRECTED-CUT | 0.5 | 0.644 | 0.6507 |
| MAX-$k$-DIRECTED-UNCUT | 0.5 | 0.811 | 0.8164 |
| MAX-$k$-DENSE-SUBGRAPH | 0.2 | 0.2008 | 0.2664 |
| MAX-$k$-VERTEX-COVER | 0.6 | 0.8453 | 0.8784 |

Table 1: Examples for the improved approximation factors

## A simple deterministic polynomial time algorithms for 2-SAT and an improved exponential algorithm for 3-SAT

We now turn to the $k$-SAT and SAT decision problems. As peviously mentioned, 2-SAT is decideable in polynomial time while $k$-SAT for $k \geq 3$ is $NP$ complete. Note however, that Max-2-SAT is $NP$-hard.

Here is a quick sketch as to how to decide a 2SAT in polynomial time. Consider a 2CNF formula $F$ in $n$ variables. We consider each clause $(x \vee y)$ as an implication $\neg x \Rightarrow y$ and equivalently $\neg y \Rightarrow x$. We then constuct a directed graph on $2n$ nodes corresponding to the $2n$ literals and edges corresponding to the inplications above;i.e. $x \Rightarrow y$ induces an edge $(x, y)$. We then claim that $F$ is satisfiable iff there does not exist a variable $x$ such that there is a directed path from $x$ to $\neg x$ and a path from $\neg x$ to $x$.

Clearly $k$-SAT (for a CNF formula with $n$ variables, $m$ clauses and at most $k$ literals/clause) can trivially be decided in time $\tilde{O}(2^n m)$. We will tend to discuss time bounds in terms of the soft $\tilde{O}$ notation which hides lower order terms. Note that the length of the formula is at least $m$ so we will ignore any dependence on $m$.

## A relatively simple deterministic algorithm for 3-SAT

Consider a 3-CNF formulai with $n$ variables and $m$ clauses. While there are any clauses with 3 lterals (for example, $x, y, z$), branch on each of the 7 possible settings for $(x, y, z)$ that can make the clause true.

In this way, we are creating a 7-ary tree where each node specifies a truth value setting on threee distinct variables. We discontinue a branch whenever we falsify a clause.

On any branch the current truth value setting can satisfy some clauses which can then be eliminated. In other clauses, one or two variables will be eliminated.

We continue to do this until there are no clauses remaining that have three literals. If any branch satisfies all clauses or if we are left with only consistent unit clauses, the given formula is saisfiable. If we are left with two contradictory unit clauses, we are also done and the given formula is unsatisfiable.

## Finishing the simple deterministic algorithm for 3-SAT

Otherwise at each branch, we have a 2-SAT formula for which we can determine satisfiability.

The depth of the truth value tree is at most $n/3$ since we were eliminating 3 distinct variables at each level.

Hence the time complexity is $O(7^{n/3} poly(m)) = 2^{log_2 7 n/3} poly(m) \approx 2^{\frac{2.81}{3} n} poly(m)) \approx \tilde{O}(1.913)^n$ which improves the exponent obtained by naively trying all $2^n$ truth values.

We note again that the encoded length of the given formula is $O(m \log n))$ so we can ignore the $poly(m)$ factor with regard to understanding what is the best possible possible exponent (assuming the time complexity is exponential in $n$).

## A modification of the previous "7-way branching" algorithm

Instead of brancing on all 7 possible satusfying truth assignments for a given clause, here is a mlodification which we can call a "3-way branching" algorithm. Here is am using lecture slides by Willam Gasarch.

Again consider a clause with 3 literals, say $x, y, z$. Then either the formual is true setting $x = \mathit{true}$, or $x = \mathit{false}, y = \mathit{true}$, or $x = \mathit{false}, y = \mathit{false}, z = \mathit{true}$.

Viewed as a recursive algorithm, we get the recurrence $T(n) = T)n - 1) + T(n - 2) + T((n - 3)$ instead of the recurrence $T(n) = 7T(n - 3)$.

Aiming for $T(n)$ roughly equal to $\alpha^n$ for some $\alpha$, we would have $\alpha^3 = \alpha^2 + \alpha + 1$ yielding $T(n) \approx O((1.84)^n$.

And some tweaking of this 3-way branching algorithm leads to an algorithm with time compleixty $O((1.618)^n)$.

## The random walk algorithm for 2-Sat

- Although there is a deterministic polynomial time algorithm for 2-Sat, it is worth considering an interesting randomized algorithm for 2-SAT which will motivate a randomized algorithm for 3-SAT (and $k$-SAT for any fixed $k$).
- The randomized algorithm for 2-SAT (due to Papadimitriou [1991]) is based on a random walk on the line graph with nodes $\{0, 1, \ldots, n\}$. We view being on node $i$ as having a truth assignment $\tau$ that is Hamming distance $i$ from some fixed (unknown) satisfying assignment $\tau^*$ if such an assignment exists (i.e. $F$ is satisfiable).
- Start with an arbitrary truth assignment $\tau$ and if $F(\tau)$ is true then we are done; else find an arbitrary unsatisfied clause $C$ and randomly choose one of the two variables $x_i$ occurring in $C$ and now change $\tau$ to $\tau'$ by setting $\tau'(x_i) = 1 - \tau(x_i)$.

## The expected time to reach a satisfying assignment

- When we randomly select one the the two literals in $C$ and complement it, we are getting close to $\tau^*$ (i.e. moving one edge closer to node 0 on the line) with probability at least $\frac{1}{2}$. (If it turns out that both literal values disagree with $\tau^*$, then we are getting closer to $\tau^*$ with probability $= 1$.)
- As we are proceeding in this random walk we might encounter another satisfying assignment which is all the better.
- It remains to bound the expected time to reach node 0 in a random walk on the line where on each random step, the distance to node 0 is reduced by 1 with probability at least $\frac{1}{2}$ and otherwise increased by 1 (but never exceeding distance $n$). This perhaps biased random walk is at least as good as the case where we randomly increase or decrease the distance by 1 with probability equal to $\frac{1}{2}$.

**Claim:**

The expected time to hit node 0 is at most $2n^2$.

- To prove the claim one needs some basic facts about Markov chains.

## The basics of finite Markov chains

- A finite Markov chain $M$ is a discrete-time random process defined over a set of states $S$ and a matrix $P = \{P_{ij}\}$ of transition probabilities.
- Denote by $X_t$ the state of the Markov chain at time $t$. It is a memoryless process in that the future behavior of a Markov chain depends only on its current state: $Prob[X_{t+1} = j | X_t = i] = P_{ij}$ and hence $Prob[X_{t+1} = j] = \sum_i Prob[X_{t+1} = j | X_t = i] Prob[X_t = i]$.
- Given an initial state $i$, denote by $r_{ij}^t$ the probability that the first time the process reaches state $j$ occurs at time $t$; $r_{ij}^t = Pr[X_t = j \text{ and } X_s \neq j \text{ for } 1 \leq s \leq t-1 | X_0 = i]$
- Let $f_{ij}$ the probability that state $j$ is reachable from initial state $i$; $f_{ij} = \sum_{t>0} r_{ij}^t$.
- Denote by $h_{ij}$ the expected number of steps to reach state $j$ starting from state $i$ (hitting time); that is, $h_{ij} = \sum_{t>0} t \cdot r_{ij}^t$
- Finally, the *commute time* $c_{ij}$ is the expected number of steps to reach state $j$ starting from state $i$, and then return to $i$ from $j$; $c_{ij} = h_{ij} + h_{ji}$

## Stationary distributions

- Define $\mathbf{q}^t = (q_1^t, q_2^t, \ldots, q_n^t)$, the state probability vector (the distribution of the chain at time $t$), as the row vector whose $i$-th component is the probability that the Markov chain is in state $i$ at time $t$.
- A distribution $\pi$ is a stationary distribution for a Markov chain with transition matrix $P$ if $\pi = \pi P$.
- Define the underlying directed graph of a Markov chain as follows: each vertex in the graph corresponds to a state of the Markov chain and there is a directed edge from vertex $i$ to vertex $j$ iff $P_{ij} > 0$. A Markov chain is *irreducible* if its underlying graph consists of a single strongly connected component. We end these preliminary concepts by the following theorem.

**Theorem: Existence of a stationary distribution**

For any finite, irreducible and aperiodic Markov chain,

**(i)** There exists a *unique* stationary distribution $\pi$.

**(ii)** For all states $i$, $h_{ii} < \infty$, and $h_{ii} = 1/\pi_i$.

## Back to random walks on graphs

- 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. We can use the Markov inequality to obtain the probability $\frac{1}{c}$ of not finding a satisfying assignment within $c \cdot 2n^2$.

## 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 could 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 previously stated running time of $\tilde{O}(2^{\frac{\log_2 7}{3} n}) \approx 2^{.94n}$, 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 say 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 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 $p \approx [(1/2)(k/k-1)]^n$. (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 $(\frac{1}{e})^c$.

## The analysis of the claim for 3SAT

Let's assume we have one satisfying assignment $x^*$ and that this satisfying solution is Hamming distance $u$ from our starting assignment $x$. The probability of choosing this satisfying assgnment $x^*$ is $2^{-n}\binom{n}{u}$.

The probability that in the first $3u$ steps, at least $2u$ are in te right direction is

$\binom{3u}{2u}(1/3)^{2u})(2/3)^{u}) = \binom{3u}{u}(1/3)^{2u}(2/3)^{u}$

Using Stirling's approximation we have (for $u \geq 3$)
$\binom{3u}{u} \geq \frac{1}{\sqrt{5u}}\frac{3^{3u}}{2^{2u}}$
Then the probability of hitting the satisfying formula is at least

$\sum_{u=0}^{n} 2^{-n}\binom{n}{u}\binom{3u}{u}(1/3)^{2u}(2/3)^{u}$
$\geq \frac{1}{\sqrt{5n}}2^{-n}\sum_{u=0}^{n}\binom{n}{u}\frac{3^{3u}}{2^{2u}}\frac{1}{3^{2u}}\frac{2^{u}}{3^{u}}$
$= \frac{1}{\sqrt{5n}}2^{-n}(1+\frac{1}{2})^{n} = \frac{1}{\sqrt{5u}}(\frac{3}{4})^{n}$

## The ETH and SETH assumptions

The $P \neq NP$ hypothesis provides evidence for a large class of problems that are not in polynomial time (assuming the hyprothesis is true).

In a similar way, the goal of *fine-grained complexity* is to try to identify a more precise time bounds for problems known to be in polynomial time.

To do so (since we do not presently have the ability to prove unconditional results), we rely on different assumptions.

*The Exponential Time Hypothesis (ETH)*; $\exists \delta$ : 3-SAT is not decideable in time $O(2^{(1+\delta)n}$.

*The Strong Exponential Time Hypothesis (SETH)*; $\forall \gamma < 1$: SAT is not decideable in time $O(2^{\gamma n})$.

## Sublinear time and sublinear space algorithms

We consider two contexts in which randomization is usually 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. 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) 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 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 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 $1/2$.