Lecture 3 — 7th of October

Lecturer: Frederic Magniez *Scribe: Marie-Sklaerder Vie and Axel de Perthuis*

3.1 *st*-connectivity

Undirected *st*-onnectivity (USTCON) is the decision problem asking whether two vertices $(s,t) \in V^2$ in an undirected graph $G = (V, E)$ are connected by a path. Let $|V| = n$ and $|E| = m$. Note that if we assume the graph is connected we have $n - 1 \leq m \leq \frac{n(n-1)}{2}$.

The deterministic depth-first search (DFS) and breadth-search first (BFS) algorithms can solve this problem in linear time complexity $O(m+n) = O(m)$ because in the worst case scenario they will visit every edge and vertex. Their space complexity is $O(n)$ as the algorithms need to store their past and future search paths in the graph. The space complexity of these algorithms can be problematic for very large graphs (for example, the Internet graph). Therefore, we wish to find an algorithm with a smaller space complexity.

Definition 3.1. L *(logarithmic-space, also known as* LSPACE*) is the complexity class containing decision problems which can be solved by a deterministic Turing machine using a logarithmic amount of memory space.*

Definition 3.2. NL *(non-deterministic logarithmic-space, also known as* NSPACE*) is the complexity class containing decision problems which can be solved by a non-deterministic Turing machine using a logarithmic amount of memory space.*

Definition 3.3. RL (randomized logarithmic-space), *sometimes called* RLP (randomized logarithmic-space polynomial-time), *is the complexity class of computational complexity theory problems solvable in logarithmic space and polynomial time with probabilistic Turing machines with one-sided error.*

Note that L $\subseteq$ RL $\subseteq$ NL. Considering the space complexity of the DFS and BFS algorithms we know that USTCON $\in$ NL. In fact, it has been shown in 2005 by Reingold that USTCON $\in$ L. However, here we will only prove the following theorem :

Theorem 3.4. *USTCON $\in$ RL.*

To do so we consider a randomised algorithm.

Algorithm 1 USTCON Las Vegas Algorithm

Require: $s, t \in V$
Ensure: boolean indicating whether there exists a path from s to t
  $u \leftarrow s$
  while $u \neq t$ **do**
    $v \leftarrow$ random element from $\{v \mid (u, v) \in E\}$
    $u \leftarrow v$
  end while
  return true

The space complexity of this algorithm is $O(\log n)$ because it suffices to keep the current vertex in memory, which requires a maximum of $\log_2(n)$ bits. If s and t are not connected then the algorithm never terminates. If there exists a path from s to t in G then the average number of steps the algorithm requires is inferior to the cover time $C(G, u)$, which is defined as the average number of steps it takes to visit every vertex in the graph, starting at u.

Theorem 3.5. *Define $C(G)$ as $\max_{u \in V} C(G, u)$. We have $C(G) \leq 4|V||E| = 4nm$*

This theorem gives us a Monte-Carlo algorithm by terminating the Las Vegas procedure after $8knm$ steps. An iterator has to be kept in memory but the space complexity is still $O(\log n)$. Markov's inequality insures that this algorithm has a one-sided error of 2^{-k}.

Proof: To prove this we shall consider the random walk on a graph as a Markov chain with a state space V and a transition matrix P where

$$P_{ij} = \begin{cases} \frac{\omega_{ij}}{w(i)} & (i,j) \in E \text{ where } w(i) = \sum_{k \in neighbors(i)} \omega_{ik} \\ 0 & \text{otherwise} \end{cases}$$

ω_{ij} represents the weight of vertex $(i, j) \in E$, i.e. the tendency to go from i to j and conversaly. ω_{ij} is normalized to be greater than one. In the simplified problem we were considering, $\omega_{ij} = 1$ for all edges.

This generalization can represent random walks, reversible Markov chains, electrical grids, etc.

Let $X_t \in V$ be the position of the random walk at step $t \in \mathbb{N}$.

For $u, v \in G$ we define the hitting time as the expected time it takes for a random walk on G starting at u to reach v :

$$h_{u,v} = \mathbb{E}(\min\{t \in \mathbb{N} \mid X_t = v, X_0 = u\})$$

Lemma 3.6.

$$\forall (i,j) \in E, \ \ h_{i,j} \leq 2W$$

where $2W$ is the sum total of the masses of all edges :

$$W = \frac{1}{2} \sum_{(i,j) \in E} \omega_{ij} \qquad (\text{ where } (\mathrm{i},\mathrm{j}) \neq (\mathrm{j},\mathrm{i}) \ )$$

In order to prove this lemma, we need to consider the stationary distribution of our Markovian process.

Since all states $v \in V$ are positive recurrent (every vertex will be visited an infinite number of times, and the expected time to do so is finite) there exists a unique stationary distribution, described by a probability vector $\pi = (\pi_1, \ldots, \pi_n)$ such that $\pi_i = \sum_{(i,j)\in E} \pi_j P_{ji}$ i.e. $\pi P = \pi$.

We can easily check that $\pi_i = \frac{w(i)}{2W}$ is a solution.

The return time of u, defined as $h_{u,u}$, is actually related to the stationary distribution π_u and satisfies

$$h_{u,u} = \frac{1}{\pi_u} = \frac{2W}{w(u)}$$

The intuition behind this lies in the probability of $X_t = u$ being constant, and hence resembling a series of Bernoulli trials (biased coin tosses). The expected number of steps needed to get one success is then given by the expected value of the geometric distribution, which is $\frac{1}{p}$.

We can establish an upper bound on the hitting time $h_{u,v}$ where $(u, v) \in E$.

$$\begin{aligned} h_{v,v} &= 1 + \frac{1}{w(v)} \sum_{k \in neighbors(v)} h_{k,v}\omega_{vk} \\ &\geq 1 + \frac{1}{w(v)} h_{u,v} \\ h_{u,v} &\leq \left(\frac{2W}{w(v)} - 1\right) w(v) \\ &\leq 2W \end{aligned}$$

Now fix $u \in V$. By performing a depth-first search of graph G starting a u we obtain a spanning tree of G which has n vertices and $n-1$ edges. We can construct a tour $T = (u_1 = u, u_2, \ldots, u_N = u)$ of the tree such that $(u_i, u_{i+1}) \in E$ and so that it covers all the vertices of G. This tour passes each edge of the spanning tree twice, therefore $N \leq 2\,(n-1)$.

Note that $C(G)$ is less than or equal to the average time it takes to travel from u_1 to u_2, then from u_2 to u_3, etc. So $C\,(G) \leq h_{u_1,u_2} + \ldots + h_{u_{N-1},u_N}$. For all i we have $h_{u_i,u_{i+1}} \leq 2W$, and thus

$$C(G) \leq (N-1) \cdot (2W) < 4nW$$

□

Thus the theorem is proved, considering that $W = m$. The upper-bound on the expected time to go from a node to another can actually be divided by two, since there exists a path of length at most n between two connected nodes.

3.1.1 Examples

Page rank

The internet is a directed graph (V, E) , with each page pointing to other pages through hyperlinks. Internet navigation can therefore be represented as a random walk on a directed

graph. If there are no links on a page, the next page is chosen uniformly from the whole internet. To represent the transition matrix, the following quantities are introduced :

$$\begin{cases} \forall\,(i,j) \in V \times V\ ,\ \omega_{ij} = 1 \text{ if } (i,j) \in E\ ,\ 0 \text{ otherwise} \\ d_{out}(i) = \sum_{j \in V} \omega_{ij} \\ d_{in}(i) = \sum_{j \in V} \omega_{ji} \\ D = \sum_{i \in V} d_{out}(i) = \sum_{i \in V} d_{int}(i) = \frac{1}{2} \sum_{(i,j) \in E} \omega_{ij} \end{cases}$$

The transition matrix is as follows :

$$P_{ij} = \begin{cases} \frac{1}{|V|} \text{ if } d_{out}(i) = 0 \\ \frac{\omega_{ij}}{d_{out}(i)} \text{ otherwise} \end{cases}$$

This Markov chain doesn't actually necessarily have a stationnary distribution, for example if the graph is a cycle. Otherwise the stationnary distribution is $\pi_i = \frac{d_{in}(i)}{2D}$. Once calculated, the stationnary distribution gives a good index of the importance of each page. Given the size of internet, the only way to calculate this is to use an ergodic theorem, which states that any random walk converges in distribution to the stationnary distribution. This process can be slow, but can be quickened by replacing P by $P' = pP + (1-p)J$ where $J_{ij} = \frac{1}{|V|}$. This modification does not affect the stationnary distribution, but can quicken the convergence process (typical values : $p = 0.85$, 200 iterations).

Linear Graph

Let $V = \{1, \ldots, n\}$ and $E = \{(i, i+1) \mid i \in \{1, \ldots, n-1\}\}$. By theorem 3.5 we have $C(G) \leq 4\,(n-1)\,n = O\,(n^2)$.

Complete Graph

In a complete graph, we have $C(G) \leq 4n \cdot \frac{n(n-1)}{2} = O\,(n^3)$. But in fact, we can prove a tighter bound of $C(G) \sim n \log n$.

Proof: In a complete graph the cover time is closely related to the coupon collector's problem. Let τ_i denote the first step at which i vertices have been visited. The number of steps it takes to reach a new vertex is

$$\tau_{i+1} - \tau_i = \frac{n-i}{n-1}$$

Since these events are independent we have

$$\mathbb{E}\,(\tau_{i+1} - \tau_i) = \frac{n-1}{n-i}$$

and we can use the approximation of the harmonic series by the natural logarithm to show that

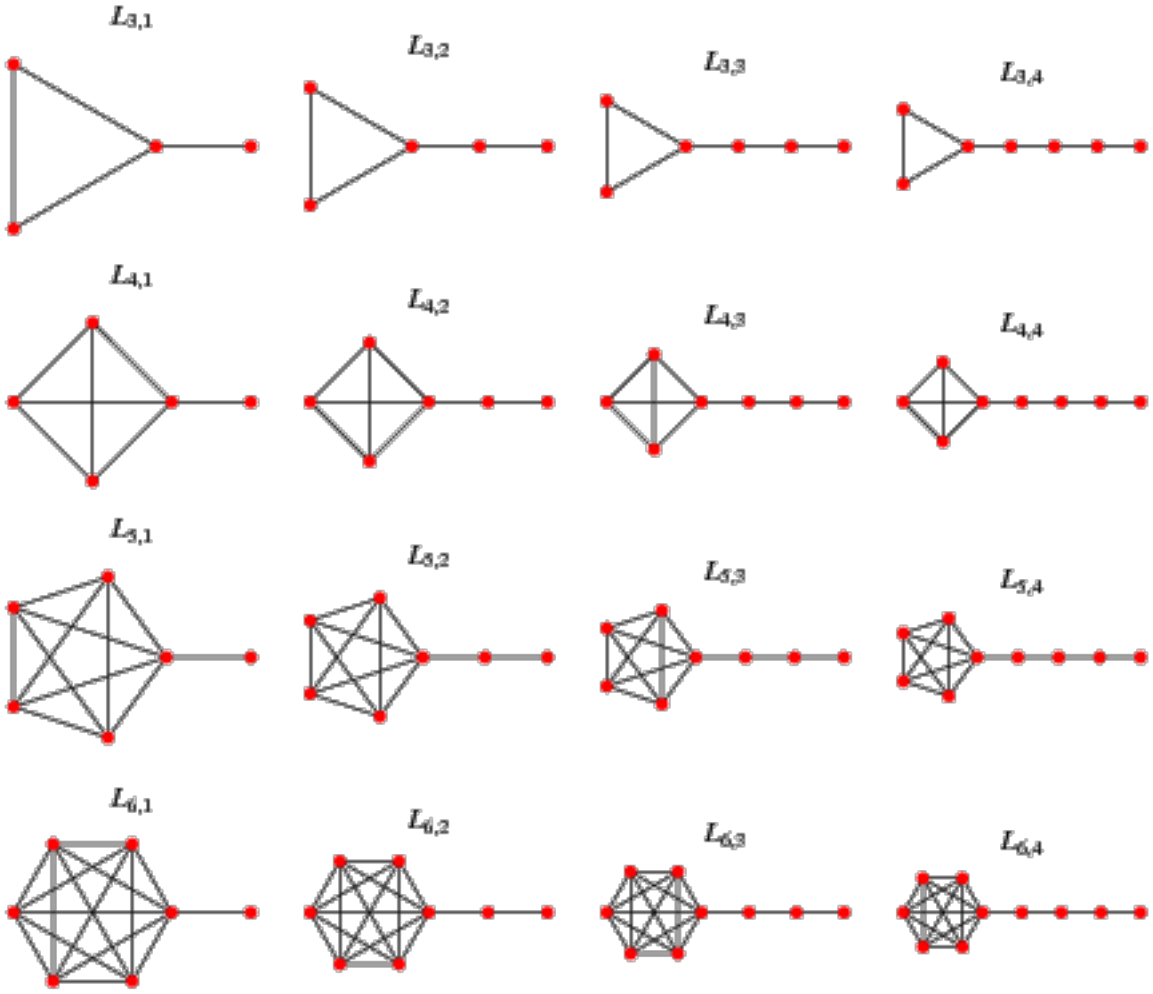
$$\mathbb{E}\,(\tau_n) = \mathbb{E}\,(\tau_1) + \sum_{i=1}^{n-1} \mathbb{E}\,(\tau_{i+1} - \tau_i) = 1 + \sum_{i=1}^{n-1} \frac{n-1}{n-i} = 1 + (n-1) \sum_{i=1}^{n-1} \frac{1}{i} \approx n \log n \text{ as } n \to \infty$$

□

Lollipop Graph

Lollipop graph is a graph as the conjunction of a linear graph (with $\frac{n}{2}$ vertices) and a complete graph (also with $\frac{n}{2}$ vertices). We deduce from the last section that : $C(G) \leq O(n^3)$.

The following graph is examples of Lollipop graphs.



Let s and t be left-most and right-most vertices of the linear graph. We can show that $h(s,t) = O(n^3)$ and $h(t,s) = O(n^2)$, which shows an interesting property of asymmetry in this graph.

3.2 Randomized algorithm for satisfiability : SAT

Definition 3.7. *Let $X_1, X_2, ..., X_n$ be $n \geq 1$ logic variables. A* literal *l is of the form X_i or $\overline{X_i}$ i.e. a literal is either a variable or the negation of a variable. A* clause *C is a disjunction of literals, for example $C = X_1 \vee X_2 \vee \overline{X_3}$. A clause C containing at most k variables is also called a k-*clause*. A* SAT formula *ϕ is of the form $\phi = C_1 \wedge C_2 \wedge ... \wedge C_m$ where C_i are clauses. A SAT formula ϕ is called a k-*SAT formula *if ϕ contains only k-clauses.*

The k-SAT is a decision problem to decide if for a given k-SAT formula there exist values of the boolean variables for which the formula is true.

We will pose the following theorems without proving them.

Theorem 3.8. *2-SAT $\in P$ and k-SAT is NP-complete for all $k \geq 3$*

Theorem 3.9. *2-SAT can be solved by a deterministic algorithm with complexity $O(n+m)$ where n is the number of variables and m is the number of clauses.*

The algorithm of the above theorem is first to construct a graph with literals of all variables, and then to check whether a variable x_i and $\overline{x_i}$ are contained in the same strongly connected component. Note that the second step can be completed within linear time using Tarjan's algorithm.

We are interested in designing a probabilistic algorithm to solve k-SAT which works for all k.

Algorithm 2 Random k-SAT algorithm

Require: a k-SAT formula ϕ, clauses $C_1, \ldots, C_m$, and literals $(\neg)X_1, \ldots, (\neg)X_n$ ▷ $n = km$
Ensure: a boolean indicating whether there exists an interpretation that satisfies ϕ
  $a \leftarrow$ any value in $\{0,1\}^n$ ▷ $a = (X_1, \ldots, X_n)$
  while $\phi(a) = 0$ **do**
    $j \leftarrow$ any integer such that $C_j(a) = 0$
    $i \leftarrow$ random integer from $\{k \mid X_k \text{ is a variable of } C_j\}$
    $X_k \leftarrow 1 - X_k$ ▷ Flip the bit of this variable in a
  end while
  return true

This algorithm will never terminate if there is no interpretation that satisfies ϕ i.e. $\phi(a) = 0$ for all a. If there is an interpretation satisfying ϕ, the algorithm will always find it, but there is no upper bound to its running time. Note that each iteration of the while-loop has a complexity of $O(mk)$ since it requires the evaluation of the entire formula, which has km literals.

Theorem 3.10. *If ϕ is 2-SAT and there exists an interpretation a such that $\phi(a) = 1$, then the average number of iterations needed to find a is $\leq 4n^2$.*

Corollary 3.11. *There exists an algorithm for 2-SAT with one-sided error 2^{-k} and running time $8kn^2$.*

Proof: For a 2-SAT problem let $s \in \{0,1\}^n$ such that $\phi(s) = 1$. Define $d(a,s) = |\{a_i \neq s_i \mid 1 \leq i \leq n\}|$. If $a = s$ then $d(a,s) = 0$ and in general $d(a,s) \in \{0, 1, ..., n\}$. Let $X_i = d(a,s)$ after i iterations.

If the algorithm has not stopped, we have

$$\mathbb{P}(X_{i+1} = n - 1 \mid X_i = n) = 1$$
$$\mathbb{P}(X_{i+1} = j - 1 \mid X_i = j) \geq \frac{1}{2} \quad \text{for} \quad 1 \leq j < n$$

The first statement is obvious because if all literals have the wrong value, changing one will always decrease the distance. For the case where $1 \leq j < n$ we can consider two cases :

- If C is a single literal, e.g. $C = \overline{X}_5 \vee \overline{X}_5$, the distance decreases with probability 1.
- If C has two literals, e.g. $C = X_2 \vee \overline{X}_7$, the distance decreases with probability $\geq \frac{1}{2}$, because at least 1 of the 2 bits is wrong, and there is a probability of $\frac{1}{2}$ that we flip the right one.

□

Note that the algorithm we have constructed is similar to a random walk on a line, where the upper bound of probability $\frac{1}{2}$ is the case where either direction is equally likely. We know that $h_{n,0}$ is upper bounded by $C(G) \leq 4n^2$.

As we did with the USTCON algorithms, we can bound the runtime of the 2-SAT algorithm by a time $8kn^2$ to construct an algorithm with a one-sided error of 2^{-k}.

Unfortunately, for 3-SAT, the distance does not necessarily decrease at each step. The state space is finite and if there is a solution it is found almost surely in a finite time. Still the algorithm can be modified to have a better convergence speed. The idea is to do the same algorithm, but only for $3n$ steps, and then to restart. It is called Walk & Restart.

Algorithm 3 Random k-SAT algorithm : Walk & Restart

Require: a k-SAT formula ϕ, clauses $C_1, \ldots, C_m$, and literals $(\neg)X_1, \ldots, (\neg)X_n$ ▷ $n = km$
Ensure: a boolean indicating that there exists an interpretation that satisfies ϕ, if there isn't, the algorithm does not terminate.

```
t ← 0, a ← any value in {0,1}^n                          ▷ a = (X_1, ..., X_n)
while φ(a) = 0 do
    while t ≤ 3n and φ(a) = 0 do
        j ← any integer such that C_j(a) = 0
        i ← random integer from {k | X_k is a variable of C_j}
        X_k ← 1 − X_k                                    ▷ Flip the bit of this variable in a
    end while
    t ← 0                                                ▷ Restart
end while
return true
```

Let us study the speed of this algorithm. Let p be the odd that the walk terminates in less than $3n$ steps. The expected number of restarts is obviously $\frac{1}{p}$.

For a 3-SAT problem let $s \in \{0,1\}^n$ such that $\phi(s) = 1$. Define $d(a,s) = |\{a_i \neq s_i \mid 1 \leq i \leq n\}|$. If $a = s$ then $d(a,s) = 0$ and in general $d(a,s) \in \{0,1,...,n\}$. Let $X_i = d(a,s)$ after i iterations.

Let p_i be the probability that a $3n$-walk terminates knowing that $X_0 = i$.

$$p = \sum_{i=1}^{n} p_i \, \mathbb{P}(X_0 = i) = \sum_{i=1}^{n} p_i \, \frac{\binom{n}{i}}{2^n}$$

When a clause a selected, one literal is different from that of s since $\phi(s) = 1$, therefore $\mathbb{P}(X_{i+1} = X_i - 1) \geq \frac{1}{3}$

$$p_j = \mathbb{P}\left(\min\left\{i,\ X_i = 0\right\} \leq 3n \mid X_0 = j\right) \geq \mathbb{P}\left(\min\left\{i,\ X_i = 0\right\} \leq 3j \mid X_0 = j\right) \geq$$

$$q_j = \mathbb{P}\left(\#\left\{i \leq 3j,\ X_{i+1} = X_i - 1\right\} = 2j,\ \#\left\{i \leq 3j,\ X_{i+1} = X_i + 1\right\} = j \mid X_0 = j\right)$$

$$q_j = \binom{3j}{j}\left(\frac{1}{3}\right)^{2j}\left(\frac{2}{3}\right)^{j}$$

$$q_j \sim \sqrt{\frac{3}{4\pi}}\frac{1}{\sqrt{j}}\left(\frac{1}{2}\right)^{j}$$

$$\sum_{i=1}^{n} p_i \frac{\binom{n}{i}}{2^n} \geq \sum_{i=1}^{n} q_i \frac{\binom{n}{i}}{2^n} \sim O(1)\sqrt{n}\left(\frac{3}{4}\right)^{n}$$

Therefore, the time complexity is $n^{O(1)}\left(\frac{4}{3}\right)^n$

As of year 2011, there is a deterministic algorithm with the same time complexity, and another randomized algorithm with time complexity $O^*(1.308^n)$

There is a version of this algorithm for k-SAT with $k \geq 4$ with time complexity $O^*\left(\left(1 + \frac{k-2}{k}\right)^n\right)$

3.2.1 Graph coloring

Consider a graph $G = (V, E)$ with n nodes, which is 3-colorable. i.e. there exists $C \in \{0, 1, 2\}^n$ such that $\forall\,(i, j) \in E,\ C_i \neq C_j$

The problem is to color this graph using two colors. To do this, the rule is that in any triangle, not all vertices are the same color. This is obviously possible since you only need to consider a 3-coloring and fuse two colors.

Consider the following algorithm

Algorithm 4 2-coloring of 3-colorable graphs

Require: a 3-colorable graph G ▷ Let T be the set of triangles of G, n the number of nodes
Ensure: a 2-coloring of G in a finite time almost surely.

```
t ← any monochromatic triangle t ∈ T, C ← any 2-coloring ∈ {0,1}^n
while there is a monochromatic triangle t ∈ T do
    t ← any monochromatic triangle t ∈ T
    i ← random vertex from t
    C_i ← 1 − C_i                          ▷ Flip the color of the vertex in C
end while
return true
```

We stated that the algorithm almost surely terminated in finite time. We now state that the expected run-time is finite.

Proof: Let C^t be the coloring after t steps of the algorithm.

Let $s \in \{0,1,2\}^n$. Many 2-coloring are naturally associated to this 3-coloring. For each vertex colored by color 2, we replace the color by 0 or 1. All such colorings are correct 2-colorings.

Let $X_t = \#\{i \in V,\ s_i \neq 2 \text{ and } s_i \neq C_i^t\}$. When a vertex i is selected at random in triangle t, there is a $\frac{1}{3}$ chance that $s_i = 2$, a $\frac{2}{3}$ chance that $s_i \neq 2$ and in that case after switching the color, there is a $\frac{1}{2}$ chance that the color was already right, and the same chance that it gets switched to the right value. Basically

$$\begin{cases} \mathbb{P}(X_{t+1} = X_t) = \frac{1}{3} \\ \mathbb{P}(X_{t+1} = X_t + 1) = \frac{1}{3} \text{ or } \frac{2}{3} \text{ if } X_t = 0 \\ \mathbb{P}(X_{t+1} = X_t - 1) = \frac{1}{3} \text{ or } \frac{2}{3} \text{ if } X_t = n \end{cases}$$

At worst, the algorithm ends with $X_t = 0$. process $\{X_t\}$ is a random walk on a line. Let E_k bet the expected time to go from k to $k-1$. $E_n = \frac{2}{3} + \frac{1}{3}E_n$ and $\forall k < n$, $E_k = \frac{1}{3} + \frac{1}{3}E_k + \frac{1}{3}(E_k + E_{k+1})$ Therefore, $E_k = (n+1-k)$, $\forall k$. So the expected time to go from n to 0 is $\frac{n(n+1)}{2}$ and therefore the expected time for completion is smaller than that.

□