Recitation 9 — Probability and Graph Contraction

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2013)

March 20, 2012

1 Announcements

- Assignment 7 is out. You will be implementing graph contraction algorithms.
- Today's recitation is about probability and graph contraction.

2 Probability Basics

Many of you have seen probability before. We'll quickly go through the basics and move on to more interesting things.

2.1 Conditional Probability

$P(A|B) = P(A \cap B)/P(B)$. This identity can be intuitively demonstrated on a Venn diagram.

2.2 Independence

Say I throw a fair six-sided dice three times. The probability of rolling an even number, then a four, then an even number again is $\frac{1}{24}$. This is because rolling an even number on a six-sided dice occurs with $\frac{1}{2}$ probability, and rolling a four occurs with $\frac{1}{6}$ probability, and $(\frac{1}{2})^2 \cdot \frac{1}{6} = \frac{1}{24}$. Now, say I draw three cards from a shuffled 52-card deck. What is the probability of drawing any card in the hearts suit, then the queen of hearts, then any hearts card again? This question is harder because the events here are not independent.

Formally, event A is independent from event B iff $P(A|B) = P(A)$; i.e. the probability of A remains the same whether or not B has occurred. From the identity $P(A|B) = P(A \cap B)/P(B)$, it follows that

$$P(A|B) = P(A) \iff P(A \cap B)/P(B) = P(A) \iff P(A \cap B)/P(A) = P(B) \iff P(B|A) = P(B)$$

so we see that event A being independent from event B also implies that B is independent from A.

Using the identity $P(A|B) = P(A \cap B)/P(B)$, the probability that events A, B, and C occur, i.e. $P(A \cap B \cap C)$, can be expressed as $P(A) \cdot P(B|A) \cdot P(C|A \cap B)$. If A, B, and C are mutually independent, then this is just $P(A) \cdot P(B) \cdot P(C)$, which is what made the first dice example so easy. (Note that mutual independence is a stronger condition than pairwise independence.) The second example with cards is hard because we're stuck with trying to figure out the probability of drawing the queen of hearts given that we've drawn some hearts card already, and then the probability of drawing another hearts given that we've drawn some hearts card and then the queen of hearts.

Keep in mind that events can be dependent even when they occur at the same time with the same probability. For example, consider a shell game, where a pea is hidden underneath one of three shells (with uniform probability). While each shell has a $\frac{1}{3}$ probability of containing the pea, the event that shell A contains the pea is not independent from the event that shell B contains the pea. At most one can contain the pea; there isn't a $\frac{1}{27}$ probability that all three shells contain the pea! In this example, the events of A and B containing the pea are mutually exclusive—$P(\text{A contains the pea}|\text{B contains the pea}) = 0$, or equivalently $P(A \cap B) = 0$.

2.3 Inclusion-Exclusion

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$. This principle can be intuitively demonstrated on a Venn diagram. Note that $P(A \cap B) = 0$ if and only if A and B are mutually exclusive, so be careful when directly adding probabilities.

2.4 Expected value

A random variable (usually a 1-dimensional number in the reals) is a variable representing the outcome of a random process. While ordinary variables are considered to have a single unknown value, random variables are considered to have all possible values, each with a particular probability. This isn't quite true, but it's good enough for 210, where we only deal with discrete probability.

The expected value of a random variable is the weighted mean of its possible values, where each value x is weighted by the probability that x is the outcome. For example, a random variable representing the outcomes of a fair six-sided dice has an expected value 3.5, which is the average of 1, 2, 3, 4, 5, and 6 when each is given equal weight. What would the expected value be for a dice that is unfair, such that even numbers are rolled with probability $\frac{2}{9}$ and odd numbers are rolled with probability $\frac{1}{9}$? You could do it the long way, "$1 \cdot \frac{1}{9} + 2 \cdot \frac{2}{9} + 3 \cdot \frac{1}{9} + \ldots$"; or you could just take the mean of $\langle 1, 3, 5, 2, 2, 4, 4, 6, 6 \rangle$.

The expected value function $\mathbf{E}$ (also known as "expectation") is a linear function, meaning that for any value c, $\mathbf{E}[c \cdot X] = c \cdot \mathbf{E}[X]$ and $\mathbf{E}[X] + \mathbf{E}[Y] = \mathbf{E}[X + Y]$. Furthermore, $\mathbf{E}[X \cdot Y] = \mathbf{E}[X]\mathbf{E}[Y]$ holds when X and Y are independent. For example, in the random process where we have n red dice and m blue dice, and will roll them and multiply the sum of the red values with the sum of the blue values, the expected value is simply $(3.5 \cdot n) \cdot (3.5 \cdot m) = 12.25 \cdot nm$. For more detail about the linearity of expectation, see the lecture slides.

2.5 Probability Bounds

Theorem 2.1. *(Markov's Inequality)* $P(|X| \geq a) \leq \dfrac{\mathbf{E}[|X|]}{a}$

Proof. Let Y be a random indicator variable that is 1 when $|X| \geq a$ and 0 otherwise.
Note that $\mathbf{E}[Y] = P(|X| \geq a)$.

Trivially, $aY \leq |X|$, because when $|X| < a$ then $Y = 0$ and the left hand side is zero; and when $|X| \geq a$, then $Y = 1$ and the left hand side is a.

Thus, $\mathbf{E}[aY] \leq \mathbf{E}[|X|] \implies a\mathbf{E}[Y] \leq \mathbf{E}[|X|] \implies P(|X| \geq a) \leq \dfrac{\mathbf{E}[|X|]}{a}$ □

The quantity $\mathbf{E}[(X - \mathbf{E}[X])^2]$ is called the *variance* of X, written $\text{Var}(X)$.

Theorem 2.2. *(Chebyshev's Inequality)* $P(|X - \mathbf{E}[X]| \geq d) \leq \dfrac{\text{Var}(X)}{d^2}$

Proof. Apply Markov's inequality to $P((X-\mathbf{E}[X])^2 \ge d^2)$, resulting in $P((X-\mathbf{E}[X])^2 \ge d^2) \le \dfrac{\mathrm{Var}(X)}{d^2}$.
$(X-\mathbf{E}[X])^2 \ge d^2$ holds if and only if $|X-\mathbf{E}[X]| \ge d$, so $P((X-\mathbf{E}[X])^2 \ge d^2) = P(|X-\mathbf{E}[X]| \ge d)$
Therefore, Chebyshev's Inequality holds. □

Note that $\mathrm{Var}(aX) = \mathbf{E}[(aX-\mathbf{E}[ax])^2] = \mathbf{E}[(aX-a\mathbf{E}[X])^2] = a^2\mathbf{E}[(X-\mathbf{E}[X])^2] = a^2\mathrm{Var}(X)$.

We will use without proof that if X and Y are independent, then $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$ (the proof is similar to the proof of linearity of expectation, but messier).

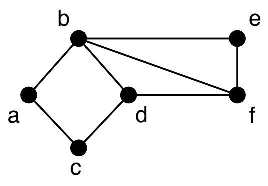
3 Maximal Independent Set (MIS)

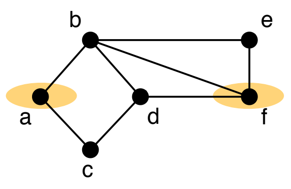
In graph theory, an *independent set* is a set of vertices from an undirected graph that have no edges between them. More formally, let a graph $G = (V, E)$ be given. We say that a set $I \subseteq V$ is an independent set if and only if $(I \times I) \cap E = \emptyset$.

For example, if vertices in a graph represent entities and edges represent conflicts between them, an independent set is a group of non-conflicting entities, which is a natural thing to want to know. This turns out to be an important substep in several parallel algorithms since it allows one to find sets of things to do in parallel that don't conflict with each other. For this purpose, it is important to select a large independent set since it will allow more things to run in parallel and presumably reduce the span of the algorithm.

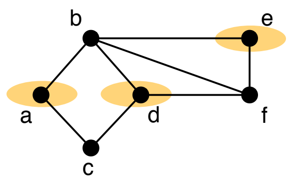
Unfortunately, the problem of finding the overall largest independent set—known as the Maximum Independent Set problem—is **NP**-hard. Its close cousin Maximal Independent Set, however, admits efficient algorithms and is a useful approximation to the harder problem.

More formally, the *Maximal Independent Set* (MIS) problem is: given an undirected graph $G = (V, E)$, find an independent set $I \subseteq V$ such that for all $v \in (V \setminus I)$, $I \cup \{v\}$ is not an independent set. Such a set I is maximal in the sense that we can't add any other vertex and keep it independent, but it easily may not be a maximum—i.e. largest—independent set in the graph.





maximal independent set



maximal and maximum independent set

For example, in the graph above, the set $\{a, d\}$ is an independent set, but not maximal because $\{a, d, e\}$ is also an independent set. On the other hand, the set $\{a, f\}$ is a maximal independent set because there's no vertex that we can add without losing independence. Note that in MIS, we are *not* interested in computing the overall-largest independent set: while maximum independent sets are maximal independent sets, maximal independent set are not necessarily maximum independent sets! Staying with the example above, $\{a, f\}$ is a maximal independent set but not a maximum independent set because $\{a, d, e\}$ is independent and larger.

3.1 Sequential MIS

Let's first think about how we would compute an MIS if we don't care about parallelism. We will start by thinking about the effect of picking a vertex v as part of our independent set I. By adding v to I, we know that none of v's neighbors can be added to I. This motivates an algorithm that picks an arbitrary vertex v from G, add v to I, and derive G' from G such that each vertex of G' is independent of I and can be picked in the next step. We have the following algorithm:

```
function seqMIS($(V,E),I$) =
if $|V| = 0$ then $I$ else
  let
      $v = \texttt{pickAnyOne}(V)$
      $V' = V \setminus (N(v) \cup \{v\})$
      $E' = E \cap (V' \times V')$
  in seqMIS($(V',E'), I \cup \{v\}$)
  end
```

In words, the algorithm proceeds in iterations until the whole graph is exhausted. Each iteration involves picking an arbitrary vertex, which is added to the independent set I, and removing the vertices v, together with v's neighbors $N(v)$, and edges incident on these vertices. Thus, each round picks a new vertex and removes exactly the vertices that can no longer be added to I, and nothing more. It is not difficult to convince ourselves that this algorithm indeed computes a maximal independent set of G. With a proper implementation (e.g., using arrays), this algorithm takes $O(m+n)$ work. Q: What algorithmic technique is this algorithm using? A: Greedy. The choice it makes is best at the time.

Homework 7 gives a parallel version of MIS using sets and tables. You need to reimplement it using array sequences and single-threaded sequences.

4 Connectivity

We're going to go over some examples from lecture in more detail today.

First, we're going to go over the LabelComponents problem: given a graph G, label each vertex so that two vertices have the same label iff they are in the same component (i.e. there is a path between them).

4.1 Graph Traversal

One way to do this is with a graph traversal algorithm such as DFS or BFS.

Q: What is the work/span of doing it with DFS?

A: $O(|E| + |V|)$ work and span.

Q: How about with BFS?

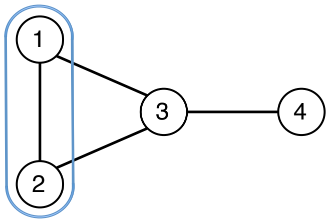
A: $O(|E| + |V|)$ work overall, and $\sum_i O(D_i \log |V_i|)$ span, where V_i is the vertex set of the ith component and D_i is the diameter of the ith component. Even though each component can be processed in parallel, we still need to visit the separate components sequentially. This can lead to linear span in the worst case.

4.2 Union Find

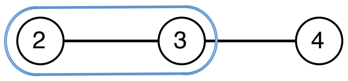
Another way is with union-find. Start with each vertex as a separate component. Then for each edge, find which components its endpoints are in and join the two components. This leads to an idea for a parallel algorithm.

```
structure UFLabel : LABEL_COMPONENTS =
struct
  structure Seq = ArraySequence
  open Seq

  type vertex = int
  type edge = vertex * vertex

  fun update (i, v) S =
      tabulate (fn j => if j = i then v else nth S j) (length S)

  fun label (E, n) =
      let
        fun contract (L, (x, y)) =
            let val (x', y') = (nth L x, nth L y)
                val L' = update (x', y') L
            in
              map (nth L') L
            end
        val L = tabulate (fn i => i) n
      in
        iter contract L E
      end
end
```

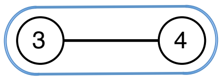
Q: What does `map (nth L') L` do? Why do we need it? This is a technique in union-find is called *path compression*, which points each vertex to its representative vertex in its component. We can demonstrate it with the following example on 4 vertices:



We start with L=⟨`1,2,3,4`⟩. If `E = ⟨(1,2), (2,3), (1,3), (3,4)⟩`, the first edge it contracts is (1, 2). After first round of contraction we get `L'`=⟨`2,2,3,4`⟩ and `L''` is the same. That is, vertices 1 and 2 are in the same component, with 2 as its representative vertex. The graph now looks like



Contracting the edge (2,3), we get `L'`=⟨`2,3,3,4`⟩. Notice that vertex 1 is still pointing to 2, but 2 is now part of component 3. Using `map (nth L') L'`, we *compress* the path 1− > 2− > 3 of length 2 to paths of length 1 and obtain L''=⟨`3,3,3,4`⟩. That gives us the following contracted graph:



Contracting the edge (1,3) has no effect, since $(x', y') = (3, 3)$. But contracting edge (3,4) we get `L'`=⟨`3,3,4,4`⟩. As before, vertices 1 and 2 are still pointing to 3, but 3 is now part of component 4. The `map` function results in the rest of the component 3 to point to 4 resulting in `L''`=⟨`4,4,4,4`⟩.

Q: What is the work/span of the above code?
A: $O(|E|^2)$ work, $O(|E|)$ span.

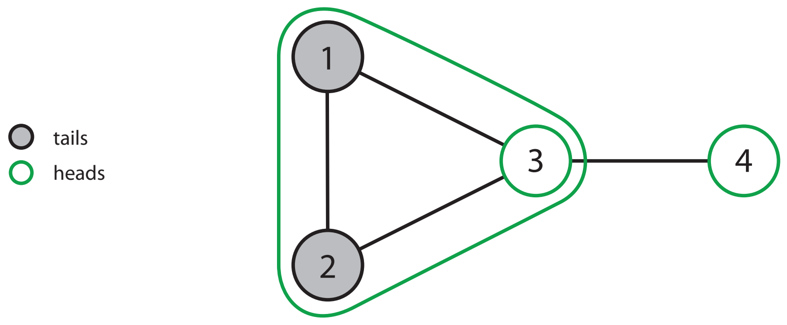
The union-find algorithm above works by contracting each edge. It is slow because it only contracts one edge at a time. Maybe we can do better with star-contraction?

4.3 Star Contraction

As in lecture, we flip a coin for each vertex, deciding if it will be the center of a star or a satellite. Then we associate each satellite with a center, and contract all of those edges. Next, we update the edges to connect to the star centers, remove new self-loops, and recurse. Except for selecting the stars to contract and updating the edges, the code is very similar to union-find above:

```
structure StarContractLabel : LABEL_COMPONENTS =
struct
  structure Seq = ArraySequence
  structure Rand = Random210
  open Seq

  type vertex = int
  type edge = vertex * vertex

  fun label (E, n) =
      let
        fun LC (L, E, seed) =
            if length E = 0 then L else
            let
              val F = Rand.flip seed n
              fun isHook (u,v) = nth F u = 0 andalso nth F v = 1
              val hooks = filter isHook E
              val L' = inject hooks L
              val L'' = map (nth L') L'
              val E' = map (fn (u, v) => (nth L' u, nth L' v)) E
              val E'' = filter (fn (u, v) => u <> v) E'
            in
              LC (L'', E'', Rand.next seed)
            end
      in
        LC (tabulate (fn i => i) n, E, Rand.fromInt 0)
      end
end
```

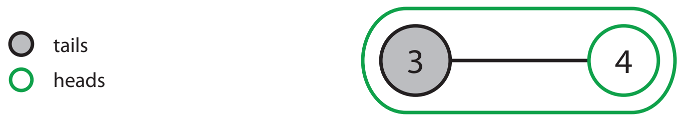
Let's run the algorithm on the example above with the coin tosses as shown:



In the first round, vertices 1 and 2 are satellites and contract to the star center, vertex 3. Vertex 4 is another star center. If `E = ⟨(1,2), (2,3), (1,3), (3,4)⟩` then after one round we have

```
hooks = ⟨(2,3), (1,3)⟩
L’ = ⟨3,3,3,4⟩
E’ = ⟨(3,3), (3,3), (3,3), (3,4)⟩
E’’ = ⟨(3,4)⟩
```

That gives us the following contracted graph:



The second and final round of `LC` contracts the graph to a single vertex and `L’’=⟨4,4,4,4⟩`.

Q: What is the work/span of the above code?
A: $O(|E| + |V|)$ work and $O(\log^2 |V|)$ span.