CME 305: Discrete Mathematics and Algorithms

Student: Laura Lyman (lymanla@stanford.edu)

HW#2 – Due at the beginning of class Thursday 02/09/17

1. **(Kleinberg Tardos 7.27)** Some of your friends with jobs out West decide they really need some extra time each day to sit in front of their laptops, and the morning commute from Woodside to Palo Alto seems like the only option. So they decide to carpool to work. Unfortunately, they all hate to drive, so they want to make sure that any carpool arrangement they agree upon is fair and doesn't overload any individual with too much driving. Some sort of simple round-robin scheme is out, because none of them goes to work every day, and so the subset of them in the car varies from day to day.

   Here's one way to define *fairness*. Let the people be labeled $S = \{p_1, \ldots, p_k\}$. We say that the *total driving obligation* of p_j over a set of days is the expected number of times that p_j would have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, suppose the carpool plan lasts for d days, and on the i^{th} day a subset $S_i \subseteq S$ of the people go to work. Then the above definition of the total driving obligation Δ_j for p_j can be written as $\Delta_j = \sum_{i:p_j \in S_i} \frac{1}{|S_i|}$. Ideally, we'd like to require that p_j drives at most Δ_j times; unfortunately, Δ_j may not be an integer.

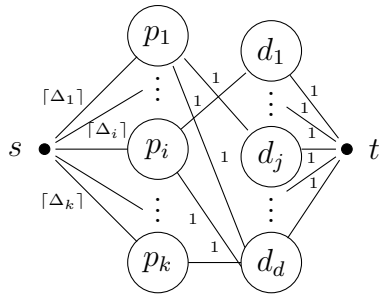
   So let's say that a *driving schedule* is a choice of a driver for each day — that is, a sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ with $p_{i_t} \in S_t$ — and that a *fair driving schedule* is one in which each p_j is chosen as the driver on at most $\lceil \Delta_j \rceil$ days.

   (a) Prove that for any sequence of sets $S_1, \ldots, S_d$, there exists a fair driving schedule.

   (b) Give an algorithm to compute a fair driving schedule with running time polynomial in k and d.

   Proof. (a) Let $p_1, \ldots, p_k$ denote the k people and $d_1, \ldots, d_d$ denote the d driving days (where d_i signifies the ith day). By above, $\Delta_j = \sum_{i, p_j \in S_i} \frac{1}{|S_i|}$ where S_i is the set of people who need to work on the ith day.

   From here, construct a flow network. We have nodes $p_1, \ldots p_k$ for all the people and $d_1, \ldots, d_d$ for all of the driving days. Then $(p_i, d_j) \in E \Leftrightarrow p_i \in S_j \Leftrightarrow$ person i can drive on the jth day. The capacity for each of these edges is 1 to indicate that each p_i can only offer one person (itself) per driving day. Then add edges $(s, p_i) \in E$ for all $i \in \{1, \ldots, k\}$ such that (s, p_i) has capacity $\lceil \Delta_i \rceil$, indicating that each p_i can drive at most $\lceil \Delta_i \rceil$ days by the fair driving restriction. Finally, we add edges $(d_j, t) \in E$ for all $j \in \{1, \ldots, d\}$ with capacity 1 to indicate that we can count each day at most once when totaling the number of days that are covered by a driving schedule. Since the maximum flow f will count the number of covered driving days, and there are d days total, $f \leq d$. When $f = d$, we have found a driving schedule such that all d days are accounted for given the restrictions on each driver's schedule and the fairness constraint; furthermore, this f is the maximum flow since $f \leq d$. The net result is a flow network of the following form:



Let $S_1, \ldots, S_d$ be an arbitrary sequence of sets such that $S_i \subseteq \{p_1, \ldots, p_k\}$. To show $S_1, \ldots, S_d$ has a fair driving schedule, we provide a specific schedule and demonstrate that it induces a maximum flow $f = d$. First, since each $|S_i| \geq 1$, $\frac{1}{|S_i|} \leq 1$. (If $|S_i| = 0$ then day i would not be included in the set of d days that need to be covered, since nobody would need to commute on the ith day.) For driver p_i, send flow $\frac{1}{|S_j|}$ to each d_j such that $(p_i, d_j) \in E$. This is possible since $\frac{1}{|S_j|}$ is at most 1 (thus meets the capacity restriction of 1 on each edge) and $\sum_{j,p_i \in S_j} \frac{1}{|S_j|} = \Delta_i \leq \lceil \Delta_i \rceil$, meaning each p_i has enough available flow from s to send flow on all of these edges. Then let each (s, p_i) have incoming flow $\sum_{j,p_i \in S_j} \frac{1}{|S_j|}$ so that the inward and outward flow on each p_i is balanced.

Now, the amount of flow reaching each d_j is *exactly* 1; for example, if 3 people need to commute on day d_j, then $|S_j| = 3$ and there are 3 incoming edges with flow $\frac{1}{3}$. In general, if $|S_j| = \ell$, then there will be ℓ incoming edges to d_j all with flow $\frac{1}{\ell}$, meaning the total flow to d_j will be 1. Then the flow from each d_j to t will be 1 (so flow is balanced), which meets the capacity restriction on these edges. Eeach d_j is outputting 1 unit of flow to t and there are d total days, so the flow from this schedule will be equal to d; since $f \leq d$, this is the maximum flow. Therefore, each day is covered by this maximum flow, so there exists a fair driving schedule on d days.

(b) Ford-Fulkerson will return the maximum flow of the flow network above along with the saturated edges corresponding to this flow. These saturated edges (that is, the saturated (p_i, d_j) edges) will indicate on which days each person should drive in the fair driving schedule. Hence, the algorithm to produce a fair driving schedule is simply to run Ford-Fulkerson on the flow network and examine the saturated edges; we know the maximum flow will be d since there exists a fair driving schedule by part (a).

The runtime of Ford-Fulkerson is $O(mn)$. In this case, there are at most kd edges in the middle (which would indicate that each person needs to commute every day). There are k edges from s and d edges going to t, meaning the total number of edges is $O(kd+k+d)$. The number of nodes is $k+d+2 = O(k+d)$. Thus, Ford-Fulkerson returns the saturated edges in $O((kd + k + d)(k + d))$ time. Since this runtime is polynomial with respect to k and d, we have produced a polynomial time algorithm to compute a fair driving schedule. □

2. Recall Karger's algorithm for the global min-cut problem. In this problem we modify the algorithm to improve its running time.

(a) Prove that if we stop the original Karger's algorithm when the remaining number of vertices is

$$\max\left\{\lceil 1+n/\sqrt{2}\rceil, 2\right\},$$

the probability that we have contracted an edge in the min-cut is less than $1/2$. Lets call this procedure *Partial Karger*.

(b) Now suppose we apply *Partial Karger* to two copies of G to produce graphs G_1 and G_2. We then recursively apply these steps to G_1 and G_2 and so on until each recursive call returns a graph on two vertices. If $r(n)$ is the running time of this process as a function of the number of vertices n of G, derive a recursive equation for $r(n)$ and solve it to obtain an explicit expression for the running time (you may use $O(\cdot)$ notation to simplify your recursive equation).

(c) Show that the algorithm in part (b) produces $O(n^2)$ contracted graphs on two vertices each. Prove that the probability that at least one of them contains a global min-cut is at least $1/\log(n)$ up to a multiplicative constant.

Hint: Think of the recursion as a binary tree with paths leading to the $O(n^2)$ leaves representing the two-vertex contracted graphs.

(d) Compare the running time of the above algorithm to Karger's original given the same probability of failure.

Proof. (a) Let S be a minimum cut. Let X_i be the event that an edge of this cut is contracted on the ith step of Karger's algorithm. Suppose the algorithm is run for k steps. Then following the calculations presented explicitly in Problem (4), we have the telescoping product

$$\begin{aligned} P(\cap_{i=1}^{k}\neg X_i) &\geq (1-\tfrac{2}{n-k+1})(1-\tfrac{2}{n-k+2})\cdots(1-\tfrac{2}{n}) = \tfrac{n-k-1}{\cancel{n-k+1}}\tfrac{n-k}{\cancel{n-k+2}}\tfrac{\cancel{n-k+1}}{\cancel{n-k+3}}\tfrac{\cancel{n-k+2}}{\cancel{n-k+4}}\cdots\tfrac{\cancel{n-3}}{n-1}\tfrac{\cancel{n-2}}{n} \\ &= \tfrac{(n-k)(n-k-1)}{n(n-1)}. \end{aligned}$$

If we run Karger's algorithm until the number of remaining nodes is

$$\max\left\{\lceil 1+n/\sqrt{2}\rceil, 2\right\}$$

then we have run Karger's algorithm for $k = (n - \max\{\lceil 1+\frac{n}{\sqrt{2}}\rceil, 2\})$ steps. If $\max\{\lceil 1+\frac{n}{\sqrt{2}}\rceil, 2\} = \lceil 1+\frac{n}{\sqrt{2}}\rceil$, then we have $k = n - \lceil 1+\frac{n}{\sqrt{2}}\rceil$ and

$$\begin{aligned} P(\cap_{i=1}^{k}\neg X_i) &\geq \frac{(n-(n-\lceil 1+\frac{n}{\sqrt{2}}\rceil))(n-(n-\lceil 1+\frac{n}{\sqrt{2}}\rceil)-1)}{n(n-1)} \\ &= \frac{\lceil 1+\frac{n}{\sqrt{2}}\rceil(\lceil 1+\frac{n}{\sqrt{2}}\rceil-1)}{n(n-1)} \\ &\geq \frac{(1+\frac{n}{\sqrt{2}})(\frac{n}{\sqrt{2}})}{n(n-1)} = \frac{(1+\frac{n}{\sqrt{2}})(\frac{1}{\sqrt{2}})}{n-1} \\ &= \frac{\sqrt{2}+n}{2(n-1)}. \end{aligned}$$

Since $n-1 \leq n+\sqrt{2}$, $\frac{n+\sqrt{2}}{n-1} \geq 1$ for all $n > 1$. Hence, $\frac{\sqrt{2}+n}{2(n-1)} \geq \frac{1}{2}$. So

$$P(\cap_{i=1}^{k} \neg X_i) \geq \frac{\sqrt{2}+n}{2(n-1)} \geq \frac{1}{2}.$$

Therefore, the probability of *contracting* an edge in the minimum cut is

$$1 - P(\cap_{i=1}^{k} \neg X_k) < 1 - \frac{1}{2} = \frac{1}{2}$$

when $\max\{\lceil 1 + \frac{n}{\sqrt{2}} \rceil, 2\} = \lceil 1 + \frac{n}{\sqrt{2}} \rceil$. Now if $\max\{\lceil 1 + \frac{n}{\sqrt{2}} \rceil, 2\} = 2$, then $n = 1$ since for all $n \geq 2$ we have $\max\{\lceil 1 + \frac{n}{2} \rceil, 2\} = \lceil 1 + \frac{n}{2} \rceil$. In this case, 0 steps of Karger's algorithm can occur (since there is only 1 node) so the probability of contracting an edge in the min cut is 0 (and thus less than $\frac{1}{2}$). Hence, the probability that we have contracted an edge in minimum cut S is less than $\frac{1}{2}$.

(b) Let $r(n)$ be the running time as a function of the number of vertices. Note that contracting an edge has computational cost $O(n)$. Since Karger's algorithm contracts an edge for every step (and there are at most $O(n)$ steps), the runtime is $O(n^2)$ every time we run Partial Karger on a graph with n nodes. The overall runtime is then the runtime of Partial Karger on G (which will be $O(n^2)$) plus the cost of the algorithm running on the two produced graphs G_1 and G_2. Since Partial Karger will leave $\lceil 1 + \frac{n}{\sqrt{2}} \rceil$ nodes in G_1 and G_2, the recursive relation is

$$r(n) = 2r(\lceil 1 + \frac{n}{\sqrt{2}} \rceil) + O(n^2).$$

Since we are using big-oh notation, we can make this recurrence relation less precise for the sake of using Master's theorem. That is,

$$r(n) = 2r(\frac{n}{\sqrt{2}}) + O(n^2)$$

so the recurrence relation has the general form of $r(n) = a \cdot r(n/b) + f(n)$ for $a \geq 1, b > 1$. By Master's theorem, the runtime is then $O(n^2 \log(n))$.

(c) *Note that this solution draws influence from Karger's paper "A New Approach to the Minimum Cut Problem."*

When branching, G_1 and G_2 will have $O(\frac{n}{\sqrt{2}})$ vertices. Therefore, the height of the binary tree after continuing to branch will be $\log_{\sqrt{2}}(n)$ since there is one level in the tree per possible division of n by $\sqrt{2}$. Then the total number of contracted graphs on 2 nodes each (that is, the number of graphs on the bottom level of the binary tree) is $O(2^{\log_{\sqrt{2}}(n)}) = O(n^2)$ as desired.

Consider the graphs on the very bottom level of the binary tree. Let $p(n)$ be the probability that at least one of them contains a global minimum cut given n nodes on G. The goal is then to show $p(n) \geq \frac{1}{\log(n)}$. Let p' be the probability that a min cut edge is contracted during Partial Karger. By part (a), $p' < \frac{1}{2}$.

First we consider the probability that none of the bottom level graphs from the G_1 branch contain a global minimum cut. This occurs only if:

(1) constructing G_1 did not contract a min cut edge, but the following steps caused an edge contraction, or

(2) an edge was contracted when constructing G_1.

From part (a), we know that *not* contracting an edge in the min cut when constructing G_1 has probability $(1-p')$. An edge is then contracted in later steps with probability $(1-p(\lceil 1+\frac{n}{\sqrt{2}}\rceil))$. So by independence, case (1) occurs with probability $(1-p')\cdot(1-p(\lceil 1+\frac{n}{\sqrt{2}}\rceil))$. Again by part (a), case (2) occurs with probability $p'<\frac{1}{2}$. Hence, the probability that none of the bottom level graphs from G_1 contain a global minimum cut is:

$$p'+(1-p')(1-p(\lceil 1+\frac{n}{\sqrt{2}}\rceil))=1-(1-p')p(\lceil 1+\frac{n}{\sqrt{2}}\rceil)\leq 1-\tfrac{1}{2}p(\lceil 1+\tfrac{n}{\sqrt{2}}\rceil).$$

By symmetry, the probability that none of the bottom level graphs from the G_2 branch contain a global min cut is also at most $1-\frac{1}{2}p(\lceil 1+\frac{n}{\sqrt{2}}\rceil)$. By independence, we then have that

$$\begin{aligned}1-p(n)&\leq(1-\tfrac{1}{2}p(\lceil 1+\tfrac{n}{\sqrt{2}}\rceil))^2\\&=1-p(\lceil 1+\tfrac{n}{\sqrt{2}}\rceil)+\tfrac{1}{4}p(\lceil 1+\tfrac{n}{\sqrt{2}}\rceil)^2.\end{aligned}$$

So $p(n)\geq p(\lceil 1+\frac{n}{\sqrt{2}}\rceil)-\frac{1}{4}p(\lceil 1+\frac{n}{\sqrt{2}}\rceil)^2$. We can then proceed by strong induction to show that $p(n)\geq\frac{1}{\log(n)}$ using the inequality $p(n)\geq p(\lceil 1+\frac{n}{\sqrt{2}}\rceil)-\frac{1}{4}p(\lceil 1+\frac{n}{\sqrt{2}}\rceil)^2$. Note that $p(n)$ is the probability that at least one of the bottom level graphs has a global min cut, and these bottom level graphs are the contracted graphs on 2 vertices by definition of the recursion. Thus, we have showed that at least one of the contracted graphs on 2 vertices contains a global min cut with probability at least $\frac{1}{\log(n)}$ up to a multiplicative constant (where the constant was chosen to be 1 in this case). Since we proved at the start that the total number of contracted graphs on 2 nodes is $O(n^2)$, this completes the proof.

(d) By part (b), the runtime of Partial Karger is $O(n^2\log(n))$ per graph on n nodes. By part (c), we know the probability that at least one of the resulting contracted graphs has a global minimum cut is at least $\frac{1}{\log(n)}$. Hence, we expect to run Partial Karger $O(\log(n))$ times in order to return a graph with a global minimum cut. So the runtime in total for the algorithm in part (c) is $O(n^2\log(n))\cdot O(\log(n))=O(n^2\log(n)^2)$. However, the runtime of Karger's classic algorithm is $O(n^2m\log(n))$ given the same probability of failure (that is, when this algorithm is also run $O(\log(n))$ times). Thus, the recursive Partial Karger algorithm has much smaller runtime than that of Karger's typical algorithm given the same probability of failure. □

3. An independent set in a graph is a set of vertices with no edges connecting them. Let G be a graph with $nd/2$ edges ($d>1$), and consider the following probabilistic experiment for finding an independent set in G: delete each vertex of G (and all its incident edges) independently with probability $1-1/d$.

(a) Compute the expected number of vertices and edges that remain after the deletion process. Now imagine deleting one endpoints of each remaining edge.

(b) From this, infer that there is an independent set with at least $n/2d$ vertices in any graph with on n vertices with $nd/2$ edges.

Proof. Each vertex is deleted independently with probability $1 - \frac{1}{d}$. Therefore, a vertex remains after this deletion process with probability $\frac{1}{d}$. Then

$$\begin{aligned} E(\#\text{ of remaining vertices}) &= E(\sum_{v \in V} \mathbb{1}_{v \text{ remains}}) = \sum_{v \in V} E(\mathbb{1}_{v \text{ remains}}) \\ &= \sum_{v \in V} \frac{1}{d} \qquad \text{[vertices deleted independently]} \\ &= |V|\frac{1}{d} = \frac{n}{d}. \end{aligned}$$

Now each edge $(v, w) \in E$ remains after the deletion process $\Leftrightarrow$ both v and w were not deleted (since edges adjacent to deleted vertices are removed as well). By independence, $P(v \text{ not deleted} \cap w \text{ not deleted}) = P(v \text{ not deleted})P(w \text{ not deleted}) = \frac{1}{d}\frac{1}{d} = \frac{1}{d^2}$. Therefore

$$\begin{aligned} E(\#\text{ of remaining edges}) &= \sum_{e \in E} E(\mathbb{1}_{e \text{ remains}}) = \sum_{e \in E} P(v \text{ not deleted} \cap w \text{ not deleted}) \\ &= |E|\frac{1}{d^2} = \frac{nd}{2}\frac{1}{d^2} = \frac{n}{2d} \end{aligned}$$

completing the calculations for (a). We now consider the algorithm which performs the process of deleting vertices/edges as described in (a) and then deletes a vertex from each edge and the edge itself.

for $v \in V$ **do**
  pick v independently with probability $p = (1 - \frac{1}{d})$
  delete edges incident to v
  $V = V \setminus \{v\}$
end for
while $E \neq \emptyset$ **do**
  pick $(v, w) \in E$ uniformly at random
  pick v or w uniformly at random; without loss of generality, say v is picked
  delete edges incident to v
  $V = V \setminus \{v\}$
end while

Let S denote a vertex set returned by the algorithm; it is purely a vertex set since all edges are deleted in the final "while" loop. Suppose for contradiction that S is not independent, so $\exists\ v, w \in S$ such that $(v, w) \in E$ (where E refers to the original edge set of G). Since v, w were outputted into S, both v and w survived the deletion process, meaning $(v, w) \in E$ survived edge deletion as well. (The edge (v, w) would have only been deleted in the for/while loop if it were incident to a deleted vertex.) However, $(v, w) \in E$ must have been deleted in the while loop since the algorithm terminated. This is a contradiction. Therefore, we conclude that S is independent.

Now we compute $E(|S|)$. For every edge that survived the deletion process in the "for" loop, exactly one vertex survives in this edge. However, since there is overlap between edges (that is, two edges can share a vertex, which can then be deleted), we will overcount by assuming that the number of vertices deleted in the "while" loop is equal to the number of remaining edges; however, this is an upper bound. That is, at most one vertex is deleted per edge in the edge set at the start of the "while" loop. Since we expect $\frac{n}{d}$ vertices and $\frac{n}{2d}$ edges to be present at the start of the "while" loop (and by linearity of expectation),

$$E(|S|) = E(\#\text{ of final vertices}) \geq \frac{n}{d} - \frac{n}{2d} = \frac{n}{2d}.$$

If there existed an independent set S of size $|S| < \frac{n}{2d}$, then with nonzero probability the algorithm will output such a set; consequently, $E(|S|)$ would not be greater than or equal to $\frac{n}{2d}$. Therefore, by the probabilistic method, there exists an independent set in G with at least $\frac{n}{2d}$ vertices. □

4. Prove that a graph can only have at most $\binom{n}{2}$ different cuts that realize the global minimum cut value.

Proof. Consider Karger's minimum cut algorithm:

while $|V| > 2$ **do**
  pick $(u, v) \in E$ uniformly at random
  contract u and v
end while

Let S be a minimum cut of G. Let X_i be the event that the algorithm fails to return S on the ith iteration of the "while" loop. (Note that this is the probability of not returning S in particular; it is not the probability of not returning a minimum cut in general.) Let S have k edges across its cut; that is, there are k edges in G with one end in S and another end in $V \setminus S$. Since we assume G is unweighted, the cut size of S is just k. Furthermore, for any $v \in V$, the cut size of $\{v\}$ is $\deg(v)$, since there are $\deg(v)$ edges between $\{v\}$ and the rest of the graph. Since S has *minimum* cut size, $k \leq \deg(v)$ for all $v \in V$. Hence

$$\begin{aligned}
\frac{1}{2}\sum_{v\in V} k &\leq \frac{1}{2}\sum_{v\in V} \deg(v) && [\text{since } k \leq \deg(v)] \\
&= \frac{1}{2} 2|E| && [\text{handshake lemma}] \\
&= |E|
\end{aligned}$$

and $\frac{1}{2}\sum_{v\in V} k = \frac{k}{2}|V| = \frac{nk}{2}$. Thus, $|E| \geq \frac{nk}{2}$. We now compute

$$\begin{aligned}
P(X_1) &= P(\text{not returning } S \text{ at step 1}) \\
&= P(\text{contracting one of the } k \text{ edges across the cut of } S) \\
&= \frac{k}{|E|}
\end{aligned}$$

since edges are contracted uniformly at random. Therefore

$$P(X_1) = \frac{k}{|E|} \leq \frac{k}{nk/2} = \frac{2}{n}$$

which is *only* dependent on n. Similarly, $P(X_2 \mid \neg X_1)$ is the same exact analysis, since there are still k edges across the cut in step 2, only now there is exactly one less vertex. So $P(X_2 \mid X_1) \leq \frac{2}{n-1}$. Continuing in this manner, $P(X_i \mid \neg X_1 \cap \cdots \cap \neg X_{i-1}) \leq \frac{2}{n-i+1}$. Now the probability that the algorithm *does* return S is the probability of $\neg X_i$ for every step i. Thus

$$\begin{aligned} P(\text{algorithm returns } S) &= P\big(\neg X_1 \cap \neg X_2 \cap \cdots \cap \neg X_{n-2}\big) \\ &= P(\neg X_{n-2} \mid \neg X_1 \cap \cdots \cap \neg X_{n-3}) P(\neg X_1 \cap \cdots \cap \neg X_{n-3}) \\ &= P(\neg X_{n-2} \mid \neg X_1 \cap \cdots \cap \neg X_{n-3}) \cdots P(\neg X_2 \mid \neg X_1) P(\neg X_1) \end{aligned}$$

by repeated applications of the definition of conditional probability. Since generally $P(\neg A \mid B) = 1 - P(A \mid B)$, we have

$$P(\neg X_i \mid \neg X_1 \cap \cdots \cap \neg X_{i-1}) = 1 - P(X_i \mid \neg X_1 \cap \cdots \cap \neg X_{i-1}) \geq 1 - \tfrac{2}{n-i+1}$$

by the previous calculations. So

$$\begin{aligned} P(\text{algorithm returns } S) &\geq (1 - \tfrac{2}{3})(1 - \tfrac{2}{4}) \cdots (1 - \tfrac{2}{n-1})(1 - \tfrac{2}{n}) \\ &= \tfrac{1}{\cancel{3}} \tfrac{\cancel{2}}{\cancel{4}} \tfrac{\cancel{3}}{\cancel{5}} \tfrac{\cancel{4}}{\cancel{6}} \cdots \tfrac{\cancel{n-3}}{n-1} \tfrac{\cancel{n-2}}{n} \qquad \text{[telescoping product]} \\ &= \tfrac{2}{(n-1)n} = \tfrac{1}{\binom{n}{2}}. \end{aligned}$$

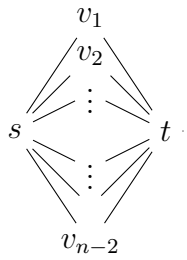
Since S was arbitrary, the probability of the algorithm returning *any* given minimum cut is at least $\frac{1}{\binom{n}{2}}$. Let ℓ be the total number of minimum cuts in G. Denote the minimum cuts by $S_1, \ldots, S_\ell$. Then by independence/disjointness,

$$P(\text{algorithm returns a minimum cut}) = P(\bigcup_{i=1}^{\ell} S_i) = \sum_{i=1}^{\ell} P(S_i) \geq \ell \tfrac{1}{\binom{n}{2}}.$$

However, since the probability of returning a minimum cut is ≤ 1, we have that $\frac{\ell}{\binom{n}{2}} \leq 1$, so $\ell \leq \binom{n}{2}$. Thus, there are at most $\binom{n}{2}$ minimum cuts. □

5. Exhibit a graph $G = (V, E)$ where there are an exponential (in $|V| = n$) number of minimum cuts between a particular pair of vertices. Do this by constructing a family of graphs parameterized on n and give a pair of vertices s, t such that there are exponentially many minimum cuts between s and t.

Proof. Consider the following graph on n vertices with $n \geq 3$:

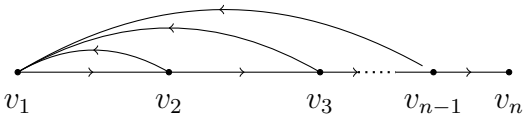


Then there are $(n-2)$ disjoint paths between s and t, meaning at least one edge from each path needs to be across the $s-t$ cut to separate s and t. Therefore, the cut size is at *minimum* $(n-2)$. For example, letting $S = \{s, v_1, \ldots, v_{n-2}\}$ so $T = V \setminus S = \{t\}$, the cut size is $(n-2)$ since each edge of the form (v_i, t) has one vertex in S and the other in T. Similarly, letting $S = \{s, v_1, \ldots, v_{n-3}\}$ and $T = \{v_{n-2}, s\}$ yields $(n-3)$ edges of the form (v_i, t) across the cut and one edge (s, v_{n-2}) across the cut as well (so now the edge (v_{n-2}, t) is an internal edge in T).

Now consider the path $s \rightsquigarrow v_1 \rightsquigarrow t$ with $s \in S, t \in T$. Since S, T partition the vertices, we can either have $v_1 \in S$ or $v_1 \in T$. These 2 choices produce 2 different cuts; since these 2 cuts have size $(n-2)$, they are minimum cuts. The other $(n-3)$ paths also produce 2 choices each for possible minimum cuts. Thus, there are 2^{n-2} total minimum cuts in this graph, meaning we have exhibited a graph on n vertices with an exponential number of nodes. □

6. Exhibit a directed graph that has cover time exponentially large in the number of nodes. Contrast this with the cover time of undirected graphs discussed in class.

Proof. Consider the directed graph drawn below on 2^n nodes:



Note that computing "effective resistance" is not sensible in the context of directed graphs. The maximum cover time will be at least $h_{i,j}$, the expected number of steps to travel from v_i to v_j. Observe that:

$$\begin{aligned}
h_{1,n} &= 1 + h_{2,n} \\
h_{2,n} &= 1 + \tfrac{1}{2}h_{1,n} + \tfrac{1}{2}h_{3,n} \\
&\vdots \\
h_{i,n} &= 1 + \tfrac{1}{2}h_{1,n} + \tfrac{1}{2}h_{i+1,n} \\
&\vdots \\
h_{n-1,n} &= 1 + \tfrac{1}{2}h_{1,n} + 0 \\
h_{n,n} &= 0.
\end{aligned}$$

Writing this as a system of linear equations,

$$\begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & \cdots & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & \cdots & \cdots & 0 \\ -\frac{1}{2} & 0 & 1 & -\frac{1}{2} & 0 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \ddots & & 0 \\ \vdots & 0 & \cdots & \cdots & \ddots & \ddots & 0 \\ -\frac{1}{2} & 0 & \cdots & \cdots & \cdots & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} h_{1,n} \\ h_{2,n} \\ \vdots \\ \vdots \\ \vdots \\ h_{n-1,n} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \end{pmatrix}$$

which reduces to

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \ddots & & & \vdots \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & & 1 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} h_{1,n} \\ h_{2,n} \\ \vdots \\ \vdots \\ \vdots \\ h_{n-1,n} \end{pmatrix} = \begin{pmatrix} 3(2^{n-2}-2) \\ \vdots \\ \vdots \\ 3(2^{n-2}-2^{i-2}) \\ \vdots \\ 3(2^{n-2}-2^{n-3}) \end{pmatrix}$$

and therefore $h_{1,n} = O(2^n)$. Since $C(G) \geq h_{1,n} = O(2^n)$, we have found a directed graph on n vertices whose cover time is exponential in n.

However, if we treat the graph as undirected, we have the bound (where $|E| = m$)

$$C(G) \leq 2m(n-1) = 2((n-1)+(n-2))(n-1) = O(n^2).$$

Thus, the directed graph has much higher cover time than the undirected graph does. Furthermore, we have for a general undirected G that

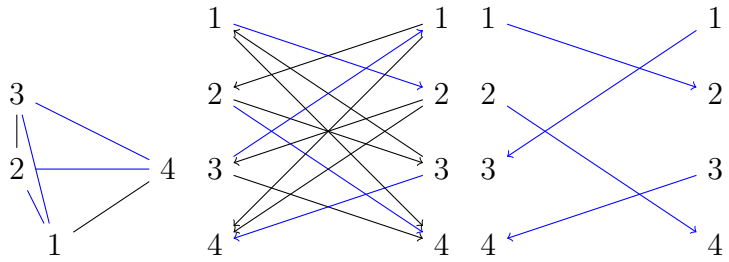
$$C(G) \leq 2m(n-1) \leq 2\binom{n}{2}(n-1) = 2\frac{n(n-1)}{2}(n-1) = O(n^3)$$

since there can be at most $\binom{n}{2}$ edges. Therefore, an undirected graph has cover time at most $O(n^3)$, but directed graphs can have exponential cover times (as seen in the example), meaning directed graphs can potentially have far worse cover times than those of undirected graphs. □

7. Given an undirected graph G, we want to find a Cycle Cover (or return None if one does not exist). Recall that a cycle cover is a set of cycles covering all nodes. Provide a polynomial time algorithm for this problem, and justify correctness.

Proof. Label the nodes $v_1, \ldots, v_n$ and make an extra copy of each. Then construct a bipartite graph with one copy of the $v_1, \ldots, v_n$ on one side and the other copy of $v_1, \ldots, v_n$ on the other side. Create directed edges (v_i, v_j) from v_i to v_j for every edge (v_i, v_j) in the original graph. (Directions are assigned to edges in the original graph arbitrarily; without loss of generality say (u, v) is from u to $v \Leftrightarrow u < v$.)

Similar to the question about permutations in HW 1, cycle covers on n nodes correspond to bijections $\{1, \ldots, n\} \to \{1, \ldots, n\}$, where the cycles are the sub-permutations of $\{1, \ldots, n\}$ that leave no node fixed (since there can be no self loops). In particular, a permutations/bijection on $\{1, \ldots, n\}$ can be represented by a perfect matching on this bipartite graph, since a perfect matching assigns every node in G exactly one corresponding node. So the desired algorithm should find a perfect matching to determine a permutation, which corresponds to a cycle cover. For example, a cycle cover of the lefthand graph below can be found by determining a perfect matching on the middle graph.

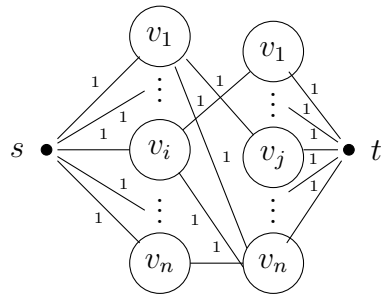


To find a perfect matching, add nodes s and t such that $(s, v_i) \in E$ for every v_i in the lefthand group of vertices and $(v_j, t) \in E$ for every v_j in the righthand group of vertices. Give every edge in the graph capacity one, as pictured below, and run Ford-Fulkerson to determine the maximum flow. If this max flow is less than n, no perfect matching exists (and thus no cycle cover exists); however, if the maximum flow is n, then the saturated edges produced by Ford-Fulkerson will be the perfect matching (and consequently these edges will be the ones comprising the cycle cover). Note that the saturated edges might contain "doubles" of certain edges (one for each direction), in which case the perfect matching will involve selecting only one edge per double (and the cycle cover will also pick one of each double, since the original graph is undirected). In summary, the algorithm is

$A, B = \{\}$
for $v_i \in V$ **do**
  make a copy of v_i
  put one v_i in A and the other in B
end for
for $v_i \in A$ **do**
  add directed edge (s, v_i) with capacity 1
end for
for $v_j \in B$ **do**
  add directed edge (v_j, t) with capacity 1
end for
for $e \in E$ **do**
  assign $e = (u, v)$ a direction
  add this directed edge between every u, v in the new graph with capacity 1
end for
run Ford-Fulkerson on new graph

if max flow $= n$ **then**
  return `True` and output saturated edges
else
  return `False`
end if

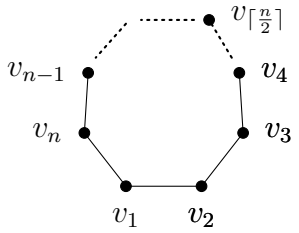
Pictorially, the graph constructed by doubling the nodes, adding $(u, v) \in E \Leftrightarrow (u, v)$ was an edge in the original graph, and adding s, t, and associated edges should have the following form.



The runtime of Ford-Fulkerson is $O(mn)$, where m in this case is $O(n^2)$ (since the number of edges is at most $n^2 + 2n$). So the runtime to output a cycle cover (or determine that no cycle cover exists) is at worst $O(n^3)$, which is polynomial time. □

8. Compute the cover time of a cycle with n vertices.

Proof. First we compute the effective resistance $R(G)$. The maximum effective resistance will occur between two vertices at opposite ends of the cycle; without loss of generality, we calculate $\Omega_{1,\lceil \frac{n}{2} \rceil} = R(G)$.



Since there are two circuits of length $O(\frac{n}{2})$ between v_1 and $v_{\lceil \frac{n}{2} \rceil}$ in parallel (where we omit the ceiling function on $\frac{n}{2}$ since we are only concerned about overall order), the computation is

$$\frac{1}{\Omega_{1,\lceil \frac{n}{2} \rceil}} = \frac{1}{n/2} + \frac{1}{n/2} = \frac{4}{n}$$

so $\Omega_{1,\lceil \frac{n}{2} \rceil} = \frac{n}{4} = O(n)$. Thus $R(G)$ is $O(n)$. Since $mR(G) \leq C(G)$ (and the number of edges $m = n$), $C(G)$ is $\Omega(n^2)$. Furthermore, since $R(G) \leq 2m(n-1) = 2n(n-1) = O(n^2)$, we have a *tight* bound on the cover time: $C(G)$ is $\Theta(n^2)$. □

9. Suppose we have a $2n \times 2n$ ($n \geq 2$) table where each cell is filled with an integer in $\{1, 2, 3, ..., 2n^2\}$. Moreover, each integer shows up exactly twice. Show that one can pick $2n$ cells that satisfy all the following conditions: (1) all the numbers written in these cells are distinct, (2) in each row exactly one cell is picked out, and (3) in each column exactly one cell is picked out.

Proof. Let $c_{i,j}$ be the cell in the ith row and jth column of the board. Consider the following algorithm to select $2n$ cells out of the board:

$S = \{\}$
uniformly at random select a permutation π of $\{1, \ldots, 2n\}$
for $i = 1, \ldots, 2n$ **do**
  $S = S \cup \{c_{i,\pi(i)}\}$
end for

Since one $c_{i,\pi(i)}$ is picked per row i, there is exactly 1 cell in every row. Since permutations are bijections, each $c_{i,\pi(i)}$ has a distinct column $\pi(i)$ and all of the columns are covered. Hence, S has $2n$ cells satisfying conditions (2) and (3). We now calculate the probability that a cell in S fail condition (1); if this probability is strictly less than 1, then by the probabilistic method, there exist $2n$ cells satisfying all of the conditions.

We know that every $k \in \{1, \ldots, 2n^2\}$ appears exactly twice on the board. Let k be arbitrary but fixed, and let the two k's appear at $c_{i,j}$ and $c_{i',j'}$. Then S will contain a pair of k's $\Leftrightarrow \pi(i) = j$ and $\pi(i') = j'$, so both $c_{i,j}$ and $c_{i',j'}$ are picked. If $i = i'$, then $P(\pi(i) = j \text{ and } \pi(i') = j')$ is zero ($i = i' \Rightarrow \pi$ must send i and i' to the same value, but if $j = j'$ then $(i, j) = (i', j')$ so k only occurs at one position, which is a contradiction). If $i \neq i'$ then

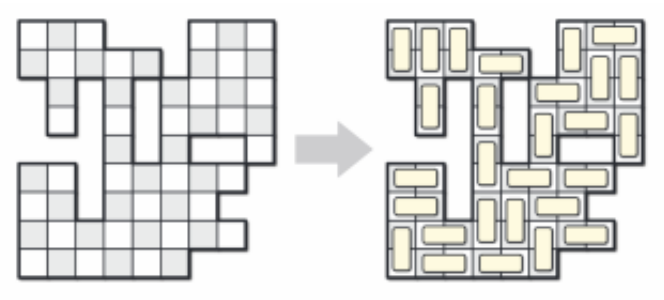
$$\begin{aligned} P(S \text{ has both } k\text{'s}) &= P(\pi(i) = j, \pi(i') = j') \\ &= \frac{\#\text{ of permutations s.t. } \pi(i) = j, \pi(i') = j'}{\text{total } \#\text{ of permutations}} \\ &= \frac{(2n-2)!}{2n!} = \frac{1}{2n(2n-1)} \end{aligned}$$

since there are $2n!$ total permutations on $\{1, \ldots, 2n\}$, $\pi(i)$ and $\pi(i')$ are fixed values, and the other $(2n-2)$ values in the permutation can be permuted in any fashion. Note that this probability is independent of $k \in \{1, \ldots, 2n^2\}$. Then

$$\begin{aligned} P(S \text{ has a double}) &= P\Big(\bigcup_{k=1}^{2n^2} S \text{ has both } k\text{'s}\Big) \\ &\leq \sum_{k=1}^{2n^2} P(S \text{ has both } k\text{'s}) \qquad \text{[union bound]} \\ &= \sum_{k=1}^{2n^2} \frac{1}{2n(2n-1)} = \frac{2n^2}{2n(2n-1)} \\ &= \frac{n}{2n-1} \end{aligned}$$

and $\frac{n}{2n-1} < 1$ for $n \geq 2$. Hence, $P(S \text{ has a double}) < 1$ for $n \geq 2$. Therefore $P(S \text{ satisfies all conditions}) = 1 - P(S \text{ has a double}) > 0$ for $n \geq 2$. We conclude by the probabilistic method (with $n \geq 2$) that there exist $2n$ cells satisfying conditions (1), (2), and (3). $\square$

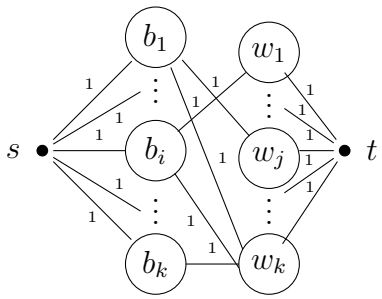
10. Say we have an $n \times n$ checkerboard. The tiles are two-colored, i.e. white and black. We delete an equal number of white and black squares from the board. Describe and analyze an algorithm to determine whether an efficient tiling of Dominoes (which are 2×1 pieces) exists, subject to the constraint that each square is covered and no domino is hanging off the board.



Your input is a two-dimensional indicator array, of size $n \times n$, whose i, j value is one if and only if the square in row i and column j has been deleted. Your output is a single bit; you do *not* have to compute the actual placement of the dominoes. In the example shown above, your algorithm should return `True`.

Proof. Covering two tiles with a domino is equivalent to matching a black and white tile. So covering the modified checkerboard with dominos is the same as pairing up all of the black and white tiles such that all of the tiles are paired and no tile is part of more than one pairing. As a graph, we can represent all the black tiles as nodes $b_1, \ldots, b_k$ and all of the white tiles as $w_1, \ldots, w_k$ (where $k = \frac{n^2}{2}$). Note that we do not add a floor/ceiling function to $\frac{n^2}{2}$, because if n^2 is not even then no tiling can exist because there is not an even number of squares.

An edge (b_i, w_j) between these two groups of nodes corresponds to a black and white tile that are next to each other in the checkerboard (and hence are possible to pair with each other). The net result is a bipartite graph. The goal given these constraints is to find a *perfect matching* in the bipartite graph, which corresponds to every tile being part of one pairing. To then find a perfect matching, add nodes s and t as done below.



We then run Ford-Fulkerson to determine the maximum flow on this graph from s to t. If there is a flow of size k, then there is a bipartite matching of size k. So a perfect matching

exists $\Leftrightarrow$ the maximum flow equals $\frac{n^2}{2}$, meaning all $2 \cdot (\frac{n^2}{2}) = n^2$ tiles are covered. Thus, the algorithm is to run Ford-Fulkerson on the above graph and return `True` if the max flow equals $\frac{n^2}{2}$ (that is, if a perfect matching exists) and `False` otherwise. To calculate the runtime, note that each tile can have at most 4 neighbors, so the total number of edges (also adding the edges with s or t) is loosely bounded above by $4 \cdot n^2 + \frac{n^2}{2} + \frac{n^2}{2} = O(n^2)$. Ford-Fulkerson runs is $O(nm)$ time, so the runtime is $O(n^2(n)) = O(n^3)$. Hence the algorithm determines whether an efficient domino tiling exists in polynomial time. $\square$