EECS 376: Foundations of Computer Science
University of Michigan, Winter 2024

Solutions to Homework 10

This homework has 8 questions, for a total of 100 points and 5 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't turn in the previous homework, then (1) state that you didn't turn it in, and (2) pick a problem that you think is particularly challenging from the previous homework, and explain the answer in your own words. You may reference the answer key, but your answer should be in your own words.

Solution:

2. **Job scheduling.**

Consider the problem of *fastest-completion job scheduling*: allocating a collection of jobs to servers so as to minimize the time until all the jobs are completed. An input is a number of identical servers n, and the times $t_1, \ldots, t_m$ that the m jobs require, i.e., t_i is how long it takes any server to complete job i. The desired output is an allocation of the jobs to the servers having minimum *completion time*, where the completion time of an allocation is the maximum, over all the servers, of the total time of the server's allocated jobs. Each job must be allocated to a single server.

It turns out that this is an NP-complete problem, but there is a polynomial-time (greedy) 2-approximation algorithm:

Solutions to Homework 10

```
1: function GreedyAllocate(n, t_1, ..., t_m)
2:     while there is an unallocated job do
3:         allocate an arbitrary unallocated job to a server that would finish its already-allocated jobs earliest
4:     return the computed allocation
```

(6 pts) (a) Suppose that there are $n = 3$ servers and $m = 7$ jobs, where $t_1 = \cdots = t_6 = 1$, and $t_7 = 3$. Determine, with justification, an optimal allocation and its completion time.

Then, consider running GreedyAllocate on this input. Show that, depending on the order in which jobs are allocated by the algorithm, the resulting allocations can have different completion times and approximation factors. Specifically, give two sequences of choices made by the algorithm that result in different completion times, and for each ordering, give its approximation ratio relative to the optimal completion time.

Solution:

One way to allocate jobs is to to allocate job 7 first and jobs 1–6 afterwards. This would yield an optimal allocation, with completion time 3. This is optimal because the jobs require a total of 9 work hours to complete, so since there are 3 servers, at least one server must work for at least 3 hours. In this case, the approximation factor is 1 because we have an optimal allocation.

Another way to allocate jobs would be to allocate jobs 1–6 first (equally among the three machines), and then job 7 last. This would yield two machines that work for 2 hours each, and another machine that works for 5 hours. In this case, the approximation factor is 5/3.

(6 pts) (b) Prove that when $m \leq n$, GreedyAllocate necessarily returns an optimal allocation.

Solution: Because $m \leq n$, the algorithm allocates each job to a server that has received zero total work so far, because there is always such a server available, and no server can finish its allocated work before time zero. Hence the algorithm obtains an overall completion time of $t^* = \max\{t_1, \ldots, t_m\}$. As shown in the next part, $t^* \leq \text{OPT}$, so the algorithm obtains the optimal completion time.

(6 pts) (c) We have handled the case $m \leq n$, so suppose that $m > n$. Letting OPT be the optimal completion time, prove that $\text{OPT} \geq \max\{t_1, t_2, \ldots, t_m\}$, and $\text{OPT} \geq \frac{1}{n} \cdot \sum_{j=1}^{m} t_j$.

Solution: A longest one of the jobs takes time $t^* = \max\{t_1, \ldots, t_m\}$ and must be assigned entirely to some server, so that server's completion time is at least t^*. Since the overall completion time is at least the finishing time of that server, the optimal completion time $\text{OPT} \geq t^* = \max\{t_1, t_2, \ldots, t_m\}$.

We now show the bound $\text{OPT} \geq \frac{1}{n} \cdot \sum_{j=1}^{m} t_j$. To see this, notice that the total time worked by *all* the servers must equal the *total* job time $T := \sum_{j=1}^{m} t_j$. Since there are n servers, at least one server must work for at least T/n time (otherwise the total time worked by all the servers would be less than T), hence $\text{OPT} \geq T/n$.

(6 pts) (d) Prove that ALG $\leq 2 \cdot$ OPT, where ALG is the completion time obtained by GreedyAllocate.

Hint: Consider a server that finishes last (i.e., it finishes at the completion time), and how the jobs were allocated before this server's final job was allocated to it.

Solution: Consider a server S that finishes its final job last, i.e., at the overall completion time. We claim that the jobs allocated to S, excluding its final one, take at most $t' := \frac{1}{n} \cdot \sum_{j=1}^{m} t_j$ time. Assume otherwise for contradiction, i.e., that S had strictly more than t' work assigned to it before getting its final job. Because the algorithm assigned that final job to S, *every* server must also have had more than t' work allocated to it. Summing over all servers, the total work allocated to all the servers would then exceed $nt' = \sum_{j=1}^{m} t_j$, the total time of all the jobs, which is a contradiction.

Next, S's final job takes at most $\max\{t_1, t_2, \ldots, t_m\}$ additional time. So, the total time S works, which by assumption is the completion time of the algorithm's entire allocation, is at most

$$\frac{1}{n} \cdot \sum_{j=1}^{m} t_j + \max\{t_1, t_2, \ldots, t_m\} .$$

By the previous part, this is at most $2 \cdot$ OPT, because each of the two terms is at most OPT, and the proof is complete.

3. **Indicator variables and linearity of expectation.**

Let $G_{n,p}$ denote a random undirected graph on n vertices, constructed as follows: for each (unordered) pair of distinct vertices $u, v \in V$, include the edge (u, v) in $G_{n,p}$ with probability p, independently of all other random choices.

(6 pts) (a) Derive the expected number of edges in $G_{n,p}$.

Solution: There are $\binom{n}{2}$ possible edges in $G_{n,p}$, one for each pair of distinct vertices. Let P be the set of such potential edges, and for each $e \in P$, let X_e be an indicator variable which is 1 if e is actually included in $G_{n,p}$, and 0 otherwise. By definition of $G_{n,p}$, each X_e has $\mathbb{E}[X_e] = \Pr[X_e = 1] = p$. The number of edges in $G_{n,p}$ is the random variable

$$X \coloneqq \sum_{e \in P} X_e ,$$

so by linearity of expectation, the expected number of edges in $G_{n,p}$ is

$$\mathbb{E}[X] = \sum_{e \in P} \mathbb{E}[X_e] = \sum_{e \in P} \Pr[X_e = 1] = \sum_{e \in P} p = p\binom{n}{2} .$$

(6 pts) (b) Derive the expected degree of any vertex in $G_{n,p}$.

Solution: Let v be an arbitrary vertex in $G_{n,p}$ and $v_1, \ldots, v_{n-1}$ be the other $n-1$ vertices. For any $1 \leq i \leq n-1$, let

$$X_i = \begin{cases} 1 & \text{if } (v, v_i) \text{ is an edge in } G_{n,p} \\ 0 & \text{otherwise.} \end{cases}$$

By definition of $G_{n,p}$, we have that $\mathbb{E}[X_i] = \Pr[X_i = 1] = p$. The degree of v is the number of edges adjacent to v, i.e., the number of vertices v_i where (v, v_i) is an edge in G, so $\deg(v)$ is the random variable

$$X = \sum_{i=1}^{n-1} X_i.$$

So, by linearity of expectation, the expected degree of v is

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \mathbb{E}[X_i] = \sum_{i=1}^{n-1} p = p(n-1).$$

Alternatively, we can derive the expected degree from our answer to the previous part. Note that in any graph $G = (V, E)$,

$$\sum_{v \in V} \deg(v) = 2 \cdot |E|$$

since any edge has two endpoints (this is sometimes known as the handshake lemma), and thus contributes 2 to the sum of the degrees of all vertices.

Letting d be the expected degree of any single vertex (all vertices have the same expected degree by symmetry), the expected value of the above sum is nd, so from the previous part we have

$$nd = 2p \cdot \binom{n}{2} \implies d = \frac{2p}{n}\binom{n}{2} = p(n-1).$$

(6 pts) (c) A *triangle* in a graph is a set of three (distinct) vertices that have an edge between every pair of them. Derive the expected number of triangles in $G_{n,p}$.

Solution: Define $N = \binom{n}{3}$ indicator random variables $X_{a,b,c}$ for distinct $a, b, c \in V$, where $X_{a,b,c} = 1$ exactly when the vertices a, b, c form a triangle in G. Then the expected number of triangles in $G_{n,p}$ is $\mathbb{E}[\sum_{i=1}^{N} X_i] = \sum_{i=1}^{N} \mathbb{E}[X_i]$. Vertices a, b, c form a triangle in G if and only if $(a, b), (b, c), (a, c) \in E$. Edges are added independently, so the probability that all three edges are added is p^3, hence $\mathbb{E}[X_{a,b,c}] = p^3$. So, the expected number of triangles is $Np^3 = \binom{n}{3}p^3$.

(6 pts) (d) Prove that the number of triangles in $G_{n,p}$ is at least n^3p^2 with probability at most $p/6$.

Solution: Let T be the number of triangles in G (so T is a non-negative random variable). Since $n, p > 0$ we have that $n^3p^2 > 0$, so by Markov's inequality,

$$\Pr[T \geq n^3p^2] \leq \frac{\binom{n}{3}p^3}{n^3p^2} \leq \frac{n^3p^3/6}{n^3p^2} \leq \frac{p}{6},$$

where the second inequality follows from the definition of $\binom{n}{3} = n(n-1)(n-2)/3! \leq n^3/6$. So, if p is fairly small (e.g., a constant or $o(1)$), then the probability that so many triangles exist in $G_{n,p}$ is similarly small.

4. **Randomized max-cut.**

In this problem, all graphs are undirected and (as usual) have *no self-loops*, i.e., there is no edge from a vertex to itself. For a cut $S \subseteq V$ in a graph $G = (V, E)$, let $C(S) \subseteq E$ denote the subset of edges "crossing" the cut, i.e., those that have exactly one endpoint in S. The size of the cut is then $|C(S)|$. Consider the following randomized algorithm that outputs a cut in a given graph $G = (V, E)$.

```
1: initialize S = ∅
2: for all v ∈ V do
3:     put v into S with probability 1/2, independently of all others
4: return S
```

(7 pts) (a) Define suitable indicator variables and use linearity of expectation to prove that *in expectation*, the above algorithm obtains a 1/2-approximation for MaxCut. That is, the expected size of the output cut is at least half the size of a maximum cut.

Solution: Let S be the random variable representing the output of the algorithm.
Let $E = \{e_1, e_2, \ldots, e_m\}$ where $m = |E|$.
For each $i = 1, \ldots, m$, let E_i be an indicator random variable that is 1 if $e_i \in C(S)$, and 0 otherwise.
Consider an arbitrary edge $e_i = (u, v)$; note that u, v are distinct because there are no self-loops.
Then because E_i is an indicator variable,

$$\begin{aligned}\mathbb{E}[E_i] &= \Pr[E_i = 1] \\ &= \Pr[u \in S \wedge v \notin S] + \Pr[u \notin S \wedge v \in S].\end{aligned}$$

Because u and v are randomly placed (or not) in S *independently* (they are distinct vertices), the above is

$$\begin{aligned}&= \Pr[u \in S] \cdot \Pr[v \notin S] + \Pr[u \notin S] \cdot \Pr[v \in S] \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \\ &= \frac{1}{2}.\end{aligned}$$

Now we can analyze the expected size of the cut. By definition of the E_i and linearity of expectation,

$$\mathbb{E}[|C(S)|] = \mathbb{E}\Big[\sum_{i=1}^{m} E_i\Big] = \sum_{i=1}^{m} \mathbb{E}[E_i] = \sum_{i=1}^{m} \frac{1}{2} = \frac{m}{2}.$$

Since no cut can have size larger than the total number of edges in the graph, we know that a maximum cut has size $\text{OPT} \leq m$.

Thus,

$$\mathbb{E}[|C(S)|] = \frac{m}{2} \geq \text{OPT}/2.$$

Therefore, this algorithm obtains a 1/2-approximation, in expectation, to undirected MaxCut.

(7 pts) (b) Prove that $\Pr[|C(S)| \geq (1-\varepsilon)|E|/2] \geq \frac{\varepsilon}{1+\varepsilon}$ for any $\varepsilon > 0$.
Notice that for $\varepsilon = 1/(2|E|)$, this is a lower bound on the probability that the number of edges crossing S is at least $|E|/2$, because the number of crossing edges is an integer.

Solution: Let N be the number of edges not crossing the cut. Then, $N = |E| - |C(S)|$ and $\mathbb{E}[N] = |E| - \mathbb{E}[|C(S)|] = |E|/2$. So, by Markov's inequality, for any $a > 0$ we have that

$$\Pr[N \geq a] \leq \frac{\mathbb{E}[N]}{a} \leq \frac{|E|}{2a}\,. \tag{1}$$

Since $|C(S)| + N = |E|$, we have that

$$\Pr[|C(S)| \geq (1-\varepsilon)|E|/2] = \Pr[N \leq (1+\varepsilon)|E|/2] = 1 - \Pr[N > (1+\varepsilon)|E|/2]\,.$$

So, by applying (1) with $a = (1+\varepsilon)|E|/2$, the above is bounded as

$$1 - \Pr[N > (1+\varepsilon)|E|/2] \geq 1 - \frac{1}{1+\varepsilon} = \frac{\varepsilon}{1+\varepsilon}\,,$$

as claimed.

Alternatively, we can immediately obtain the stated bound using the "reverse" Markov inequality, since $|C(S)| \leq |E|$. (The proof of the "reverse" Markov inequality proceeds essentially identically to what we gave above.)

(5 pts) (c) As a stepping stone to the next part, we consider the following intermediate question. Consider a probability experiment that consists of a sequence of "attempts," where each attempt succeeds with probability p, independently of all others. We keep making attempts until one succeeds, at which point the experiment terminates.

Let X denote the number of attempts until termination (including the attempt that finally succeeds). Prove that

$$\mathbb{E}[X] = p + (1-p)(1 + \mathbb{E}[X]) \ ,$$

and conclude that $\mathbb{E}[X] = 1/p$. (The distribution of X is known as the *geometric distribution* with success probability p.)

Solution:
With probability p, we succeed on the first attempt, in which case $X = 1$. With probability $1 - p$, we fail on the first attempt. In this case, observe that the whole process "starts over," but the total number of attempts is one larger (counting the failed first attempt). So, the expected number of attempts in this case is $1 + \mathbb{E}[X]$. Altogether, this implies that

$$\mathbb{E}[X] = p + (1-p)(1 + \mathbb{E}[X]) \ .$$

Solving this equation for $\mathbb{E}[X]$, we get that

$$\begin{aligned} \mathbb{E}[X] &= p + 1 - p + \mathbb{E}[X] - p\,\mathbb{E}[X] \\ \implies p\,\mathbb{E}[X] &= 1 \\ \implies \mathbb{E}[X] &= 1/p \ . \end{aligned}$$

(6 pts) (d) Suppose we repeatedly run our randomized MaxCut algorithm until we get a cut of size at least $|E|/2$. Derive an upper bound on the expected number of attempts that are needed.

Solution: Choosing $\varepsilon = 1/(2|E|)$ as suggested by the remark in part (b), we get that

$$\Pr[|C(S)| \geq |E|/2] \geq \frac{\varepsilon}{1+\varepsilon} = \frac{1}{2|E|+1} \ .$$

That is, any single run of our randomized MaxCut algorithm outputs a cut of size at least $|E|/2$ with some probability $p \geq \frac{1}{2|E|+1}$. If we repeatedly run the algorithm (with fresh random choices each time) until we get a cut of size at least $|E|/2$, then the number of attempts follows the geometric distribution with the success probability p. By the previous part, the expected number of attempts is $1/p \leq 2|E| + 1$.

(5 pts) 5. **Quicksort.**

In class, we showed that (randomized) Quicksort has expected running time $O(n \log n)$ on an array of n elements. Although the worst-case running time of the algorithm is $\Omega(n^2)$, we will prove that this happens with small probability.

Let c_1 be the constant such that the expected running time of Quicksort is at most $c_1 \cdot n \log n$ (for all large enough n). Prove that, for any constant $c_2 > 0$, the probability that it takes time at least $c_2 n^2$ time is $O(\frac{\log n}{n})$.

Solution: Let random variable T denote the running time of Quicksort (on an array of n elements). We have that $\mathbb{E}[T] = O(n \log n)$. By Markov's inequality,

$$\Pr[T \geq c_2 n^2] \leq \frac{\mathbb{E}[T]}{c_2 n^2} \leq \frac{c_1 n \log n}{c_2 n^2} = O(\frac{\log n}{n}) \, .$$

(12 pts) 6. **Random binary search trees.**

A binary search tree on distinct values is a binary tree where for any node with value x, all the nodes in its left subtree have values less than x, and all nodes in its right subtree have values greater than x. Consider the random process that generates a binary search tree on the integers $\ell, \ldots, r$ for given $\ell \leq r$, as follows.

- Choose $x \in \{\ell, \ldots, r\}$ uniformly at random and take x as the root. If $\ell = r$, output the tree consisting of just x.
- If $\ell < x$, for x's left subtree, recursively build a random binary search tree on $\ell, \ldots, x-1$.
- If $x < r$, for x's right subtree, recursively build a random binary search tree on $x+1, \ldots, r$.

For a random binary search tree on $1, \ldots, n$, prove that the expected depth of the node with any particular value is $O(\log n)$.

Hint: Use ideas from the analysis of the Quicksort algorithm from class.

Solution: Observe that this process for generating a random binary search tree is exactly the one we used to create a binary search tree in the analysis of the Quicksort algorithm: the root corresponds to the pivot (they are chosen uniformly at random from the current interval of elements), and the left/right subtrees are the ones corresponding to the recursive calls to Quicksort on the elements smaller/larger than the pivot.

Let X_{ij} be the indicator of whether i is a descendent of j in this random tree. Observe that $X_{ij} = 1$ if and only if, in the recursive process, j is chosen as a root chosen *before* all the elements between i and j (inclusive). Recall from class that this event is equivalent to the event that j has the smallest *priority* among all those elements, when we assign them random priorities. So $X_{ij} = 1$ with probability $\frac{1}{|j-i|+1}$, hence

$$\mathbb{E}[X_{ij}] = \frac{1}{|j-i|+1}.$$

Observe that the depth of i is $\sum_{j \neq i} X_{ij}$, because this counts the number of elements j that are ancestors of i. So, the expected depth of i is

$$\mathbb{E}\Big[\sum_{j \neq i} X_{ij}\Big] = \sum_{j \neq i} \mathbb{E}[X_{ij}] = \sum_{j \neq i} \frac{1}{|j-i|+1} = O(\log n)$$

using the fact that $1 + 1/2 + \cdots + 1/n = O(\log n)$ and the above summation is at most twice this sum.

(5 EC pts) 7. **Optional extra-credit question: randomized vertex-cover.**

This question considers randomized algorithms for approximating the vertex-cover problem.

(a) Consider the following probability experiment:

- There is a fixed set of k distinct items.
- The experiment proceeds in a sequence of rounds. In each round, with probability at least $1/2$ (and independently of all other rounds), some arbitrary item that was not previously chosen is chosen. (Otherwise, nothing is chosen in this round.)
- Once every item has been chosen, the experiment ends.

Let R be the random variable denoting the number of rounds until the experiment ends. Prove that $\mathbb{E}[R] \leq 2k$.

Hint: for $j = 1, \ldots, k$, let R_j be the random variable denoting the number of rounds after the $(j-1)$st chosen item and until the jth chosen item (inclusive). What is the relationship between R and the R_j? What is an upper bound on $\mathbb{E}[R_j]$?

Solution: By definition, we have $R = \sum_{j=1}^{k} R_j$. This is because the total number of rounds is the number of rounds until the first chosen item (i.e., R_1), plus the number of rounds after that until the second chosen item (i.e., R_2), etc. So, by linearity of expectation, $\mathbb{E}[R] = \sum_{j=1}^{k} \mathbb{E}[R_j]$.

It therefore suffices to upper bound each $\mathbb{E}[R_j]$. To do this, note that in each round, an item is chosen with probability at least $1/2$ (independently of all other other rounds), and R_j is the number of rounds until an item is chosen. This is equivalent to repeatedly flipping fair-or-better coins—i.e., ones with (possibly different) biases $\geq 1/2$ toward "heads"—with R_j being the number of flips until we get "heads." The expected number of flips is known to be $\mathbb{E}[R_j] \leq 1/(1/2) = 2$. (This is covered in EECS 203; for completeness, below we give a derivation using geometric series.) Summing up, we have that $\mathbb{E}[R] = \sum_{j=1}^{k} \mathbb{E}[R_j] \leq \sum_{j=1}^{k} 2 = 2k$.

To prove the above claim about $\mathbb{E}[R_j]$, first observe that for any random variable V that takes on positive integer values, when $\mathbb{E}[V]$ exists we have

$$\begin{aligned}\mathbb{E}[V] &= 1 \cdot \Pr[V = 1] + 2 \cdot \Pr[V = 2] + 3 \cdot \Pr[V = 3] + \cdots \\ &= \Pr[V \geq 1] + \Pr[V \geq 2] + \Pr[V \geq 3] + \cdots,\end{aligned}$$

because $\Pr[V \geq i] = \Pr[V = i] + \Pr[V = i + 1] + \cdots$ for any integer i.

Now notice that $\Pr[R_j \geq i] \leq 1/2^{i-1}$, because $R_j \geq i$ exactly when the first $i - 1$ rounds in question all fail to choose an item, and each round fails independently with probability $\leq 1/2$. So, by the above characterization of expectation we have

$$\mathbb{E}[R_j] \leq 1 + 1/2 + 1/4 + \cdots = 2.$$

(b) Prove that the following randomized algorithm outputs a 2-approximation, in expectation, for the minimum vertex cover problem.

Hint: let $C^* = \{v_1, \ldots, v_k\}$ be some minimum vertex cover in the input graph. Show how a run of the algorithm corresponds to a (complete or partial) run of the experiment from the previous part with the v_i as the items, and use this to bound the expected size of the output C.

```
1: function RandomSingleCover(G = (V, E))
2:     C = ∅
3:     while there is some edge e = (u, v) that is not covered by C do
4:         add exactly one of u or v to C, each with probability 1/2
5:     return C
```

Solution: Let $C^* = \{v_1, v_2, \ldots, v_k\}$ be a minimum vertex cover for the input graph $G = (V, E)$. Below we prove that a run of the algorithm corresponds to a (complete or partial) run of the probability experiment from the previous part, with the v_i as the items. Specifically, we make these two claims:

- in each loop iteration, some not-yet-chosen element of C^* is chosen and included in C with probability at least $1/2$, so C always includes all the vertices of C^* that the experiment has chosen so far (and possibly others); and
- the algorithm halts *no later* (and possibly earlier) than when the experiment ends.

Given these claims, observe that when the algorithm halts, $|C|$ is exactly the number of loop iterations that were run (because exactly one vertex is added to C in each iteration), and this is at most the number of rounds until the experiment ends. So, by the previous part, $\mathbb{E}[|C|] \leq 2k$. Since $|C^*| = k$ is a minimum-size vertex cover, we conclude that C is a 2-approximate minimum vertex cover in expectation.

Proof of above claims. We first show that the claimed invariant holds after every loop iteration, i.e., C includes all the chosen-so-far vertices in C^*. Clearly, the invariant holds at the start: $C = \emptyset$ trivially includes all the chosen vertices of C^*, because no vertices have been chosen yet.

In each loop iteration, the algorithm chooses some edge $e = (u, v)$ that is not covered by any of the vertices in C, so it is also not covered by any of the chosen-so-far vertices of C^* (because they all are in C, by the invariant). Since C^* is a vertex cover, it covers e, so at least one of u, v must be a *not-yet-chosen* vertex of C^*. Hence, just as in the experiment (and as claimed above), with probability at least $1/2$ the algorithm chooses some not-yet-chosen vertex of C^*. And because that vertex is immediately added to C, the invariant continues to hold at the end of the iteration.

(Note that is possible for an iteration to add a vertex to C that is *not* in C^*. This corresponds to a round of the experiment that does not choose anything from C^*.)

Finally, we show that the algorithm halts no later than when the experiment ends; equivalently, if the algorithm has not halted, then the corresponding experiment has not ended. Since the algorithm has not halted, there is still some edge $e = (u, v)$ that is not covered by C. Since C contains all the chosen-so-far elements of C^*, this means that e is not covered by those chosen elements either. So, since C^* is a vertex cover of the graph, not all the elements of of C^* have been chosen yet, so the experiment has not ended, as claimed.

(Note that it is possible for the algorithm to halt *strictly before* the corresponding experiment ends, i.e., that C never contains all of C^*. For example, consider a graph with just two vertices and one edge between them; then the algorithm is guaranteed to halt after just one iteration, and in that iteration it selects the vertex that is not in C^* with probability exactly $1/2$.)