We now prove that the probability that the algorithm indeed outputs a minimum cut is bounded below.

Theorem 1.14. *The algorithm* **`Contract`** *always outputs a cut. With a probability of at least $\frac{2}{n(n-1)}$, this cut is a minimum cut.*

Proof. To see that the output S is always a cut, observe that a label can never be empty, and that it cannot be V as long as there are at least two nodes. We can say something more about the multigraphs that the algorithm creates. Recall that the *degree* of a node $v \in V$ is the number of edges $e = \{x, y\} \in E$ with $v \in \{x, y\}$. For any G_i, $i \in \{0, \ldots, n-2\}$, the degree of a node $v \in V_i$ is equal to the number of edges that leave the cut that the label of v represents. In other words, the degree of v is always $|\delta(\ell(v))|$. Formally, this statement can be shown by induction. We observe that the statement is true for G_0: Then the label of v is just $\{v\}$, and the value of this cut is exactly the degree of v. Whenever we merge two vertices u and v, we keep exactly those edges that have one endpoint in the label sets $\ell(u)$ and $\ell(v)$ and the other endpoint outside. Exactly these edges are cut by $\ell(u) \cup \ell(v)$.

There can be many minimum cuts in G, but that is not guaranteed, so we don't want to consider multiple minimum cuts in our analysis. Instead we want to compare S to a fixed minimum cut. We let S^* be an arbitrary optimal solution and now always compare to this solution. Let $C^* = \delta(S^*)$ be the edges that are cut by S^*.

If the algorithm never contracts an edge from C^*, then it outputs C^* and thus succeeds. We observe that the probability that it fails in the first iteration is $|C^*|/|E_0|$. So in order to see that this probability cannot be arbitrarily high, we need to know something about the number of edges in G_0.

We have the following insight. If a vertex in G_0 has degree d, then the cut $\{v\}$ has value d as we observed already. So if S^* is a minimum cut, then all vertices have a degree of at least $|C^*|$, because otherwise, there would be a better cut than S^*. We can now use a fact about graphs. If one sums up the degrees of all nodes, then every edge is counted exactly twice, thus the sum is $2 \cdot |E|$. That implies the following:

Fact 1.15. *If every node in a graph G has degree at least d, then the graph has at least $(n \cdot d)/2$ edges.*

We have argued that every vertex in G_0 has degree $\geq |C^*|$. Thus, Fact 1.15 says that G_0 has at least $(n \cdot |C^*|)/2$ edges. We can now bound the probability that the algorithm makes a mistake in the first iteration by

$$\frac{|C^*|}{|E_{i-1}|} \leq \frac{|C^*|}{(n \cdot |C^*|)/2} = \frac{2}{n}.$$

Let A_i be the event that the algorithm contracts a good edge in iteration i, i.e. an edge that is not in C^*. We have just shown that $\mathbf{Pr}(A_1) \geq 1 - \mathbf{Pr}(\bar{A}_1) = 1 - \frac{2}{n}$. The algorithm succeeds if A_i occurs for all $i \in \{1, \ldots, n-2\}$, i.e. the algorithm always contracts an edge that is not in C^*. We have a similar situation as when we analyzed dependent runs of our polynomial tester. The success probability of the `Contract` algorithm is:

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_{n-2}) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \ldots \cdot \mathbf{Pr}(A_{n-1} \mid A_1 \cap \ldots \cap A_{n-3})$$

where we use the definition of conditional probability. If we pick good edges in the first $i-1$ iterations, then there are still $|C^*|$ bad edges in iteration i, even though the total number of edges decreased. More precisely, the probability to choose an edge from C^* if we never chose an edge from C^* before is

$$\mathbf{Pr}(\bar{A}_i \mid A_1 \cap \ldots \cap A_{i-1}) = \frac{|C^*|}{|E_{i-1}|}.$$

Now we need to show that E_{i-1} still contains enough edges. So far, we only showed that $|E_0|$ is large. However, we can show a lower bound on $|E_{i-1}|$ in a similar way. We observed in the beginning of the proof that the degree of every vertex in G_i is exactly the value of the cut that we get from the label $\ell(v)$. We know that no cut can cut less than $|C^*|$ edges. Thus, the degree of every node in G_{i-1} is *still* at least $|C^*|$. The number of nodes in G_{i-1} is $n-(i-1)$ because we contracted $i-1$ edges. Thus, we know that $|E_{i-1}| \geq |C^*| \cdot (n-i+1)/2$ if we again use Fact 1.15. So we now know that

$$\mathbf{Pr}(\bar{A}_i \mid A_1 \cap \ldots \cap A_{i-1}) = \frac{|C^*|}{|E_{i-1}|} \leq \frac{|C^*|}{|C^*| \cdot (n-i+1)/2} = \frac{2}{n-i+1}.$$

We can now compute

$$\mathbf{Pr}(A_i \mid A_1 \cap \ldots \cap A_{i-1}) = 1 - \mathbf{Pr}(\bar{A}_i \mid A_1 \cap \ldots \cap A_{i-1}) = 1 - \frac{2}{n-i+1} = \frac{n-i-1}{n-i+1}.$$

Luckily, when we multiply the different terms, we observe that a lot of terms cancel out. We get:

$$\begin{aligned}
&\mathbf{Pr}(\text{Algo } \texttt{Contract} \text{ succeeds}) \\
&\geq \mathbf{Pr}(\text{Algo } \texttt{Contract} \text{ outputs } S^*) \\
&= \mathbf{Pr}(A_1 \cap \ldots \cap A_{n-2}) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \ldots, \cdot \mathbf{Pr}(A_{n-1} \mid A_1 \cap \ldots \cap A_{n-3}) \\
&\geq \left(\frac{n-2}{n}\right) \cdot \left(\frac{n-3}{n-1}\right) \cdot \left(\frac{n-4}{n-2}\right) \cdot \ldots \cdot \left(\frac{4}{6}\right) \cdot \left(\frac{3}{5}\right) \cdot \left(\frac{2}{4}\right) \cdot \left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)}
\end{aligned}$$

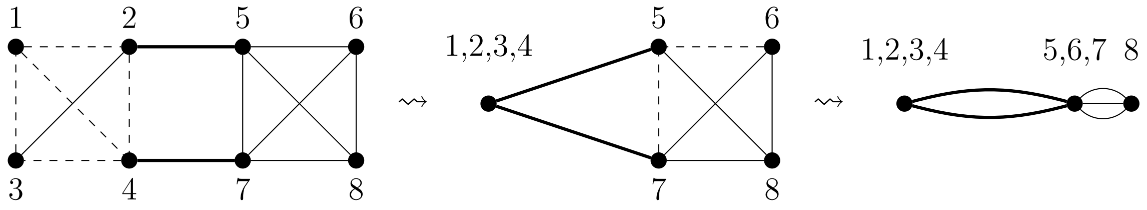
That shows the theorem. □

A success probability of roughly $1/n^2$ does not look impressive. However, we can increase the success probability by independent runs as we did before. Assume that we repeat the algorithm n^2 times, and return the best solution that we found. Then the output is not a minimum cut if no run found a minimum cut. Thus, the probability that the algorithm fails is at most

$$\prod_{i=1}^{n^2} \mathbf{Pr}(\text{run } i \text{ fails}) = \prod_{i=1}^{n^2} \left(1 - \frac{2}{n(n-1)}\right) \leq \prod_{i=1}^{n^2} \left(1 - \frac{2}{n^2}\right) = \left(\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}}\right)^2 \leq \left(\frac{1}{e}\right)^2 = \frac{1}{e^2}.$$

For the last step, we used the useful inequality $(1 - \frac{1}{x})^x < \frac{1}{e}$ which holds for $x \geq 1$. The failure probability is now $1/e^2$, which is less than 0.14. We can decrease it to any given δ by increasing the number of independent runs appropriately, as we did for the polynomial tester.

The FastCut algorithm Now we discuss an improvement of the `Contract` algorithm that we call `FastCut`. It was was published by Karger and Stein in [KS96]. The following pictures show snapshots (G_0, G_3 and G_5) of an example run of the `Contract` algorithm. The dashed edges are contracted in the steps that happen between the pictures.



An optimal solution is $\{1, 2, 3, 4\}$ which cuts the two bold edges C^*. We observe that in G_0, there are 14 edges in total, and the probability to choose one of the two bold edges is only 1/7. However, after contracting five edges not in C^*, the example run obtains G_5 with only five edges. Now the probability to contract an edge from C^* is 2/5. Additionally, the probability to get to G_5 is already small because the algorithm had to make five good choices already. The failure probability gets higher and higher the longer the `Contract` algorithm runs.

We also see this effect if look at the proof of Theorem 1.14 again. Assume that we stop the `Contract` algorithm when then graph has t nodes left, i.e. at G_{n-t}. Then the probability that the algorithm contracted no edge from C^* is

$$\begin{aligned}
&\mathbf{Pr}(A_1 \cap \ldots \cap A_{n-t}) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \ldots, \cdot \mathbf{Pr}(A_{n-t} \mid A_1 \cap \ldots \cap A_{n-t+1}) \\
&\geq \left(\frac{n-2}{n}\right) \cdot \left(\frac{n-3}{n-1}\right) \cdot \left(\frac{n-4}{n-2}\right) \cdot \ldots \cdot \left(\frac{t+2}{t+4}\right) \cdot \left(\frac{t+1}{t+3}\right) \cdot \left(\frac{t}{t+2}\right) \cdot \left(\frac{t-1}{t+1}\right) \\
&= \frac{t(t-1)}{n(n-1)}.
\end{aligned}$$

We see that the bound gets very small when the number of remaining nodes is small, e.g. for $t = 2$, which matches the original `Contract` algorithm. The key observation is that we can reduce the number of nodes significantly, for example to around $(3/4)n$, without having a high danger of failure: For $t = 1 + \lceil (3/4)n \rceil$, the success probability is still bounded below by

$$\frac{t(t-1)}{n(n-1)} \geq \frac{(t-1)^2}{n^2} \geq \frac{(3/4)^2 n^2}{n^2} = \frac{9}{16} \geq \frac{1}{2}$$

The value of t can be made a little smaller. For $t = 1 + \lceil n/\sqrt{2} \rceil \approx 0.7n$, we get

$$\frac{t(t-1)}{n(n-1)} \geq \frac{(t-1)^2}{n^2} \geq \frac{n^2}{\sqrt{2}^2 n^2} = \frac{1}{2}$$

Setting $t = 1 + \lceil n/\sqrt{2} \rceil$ will turn out beneficial for the running time, so we choose this value. The algorithm `Contract(`G, t`)` is the same as `Contract(`G`)` except that the 2 in step 2 is replaced by t, that step 6 is deleted and that the algorithm returns the current graph G instead of a cut. We just argued that the probability that `Contract(`G, t`)` never contracts an edge from C^* is at least 1/2 for $t = 1 + \lceil n/\sqrt{2} \rceil$.

The idea behind the algorithm `FastCut(`G`)` is to include the repetitions into the algorithm instead of simply repeating Karger's `Contract(`G`)` algorithm as a whole. This makes sense because the failure probability increases during the algorithm. Entangling the repetitions with the algorithm opens the possibility to repeat the more dangerous steps more often than the safer steps when the graph still has many edges. The algorithm uses `Contract(`G, t`)` as a subroutine. It uses two independent calls to this procedure to reduce G to graphs H_1 and H_2 with $1 + \lceil n/\sqrt{2} \rceil$ nodes. Then it recursively solves the minimum cut problem first on H_1 and second on H_2 by calling `FastCut(`H_1`)` and `FastCut(`H_2`)`, the better result is then output. Observe that each recursive call starts with two independent reduction steps. Thus, in the first level of the recursion, the previously computed H_1 is twice reduced to roughly 70 % of its nodes, which is 49% of the original number of nodes. The deeper we are in the recursion tree, the more independent reduced graphs are computed in different branches. When the number of nodes n falls below 7, then $1 + \lceil n/\sqrt{2} \rceil$ is no longer smaller than n (observe that $1 + \lceil 6/\sqrt{2} \rceil \geq 1 + \lceil 4.24 \rceil = 6$). When the recursion reaches this point, the algorithm simply solves the problem optimally. For graphs with less than 7 nodes, this can be done in constant time.

```
FastCut(G = (V, E))

 1. if $n := |V| \geq 7$
 2.     Set $t := 1 + \lceil n/\sqrt{2} \rceil$
 3.     $H_1 = $ Contract$(G, t)$; $H_2 = $ Contract$(G, t)$
 4.     $S_1 = $ FastCut$(H_1)$; $S_2 = $ FastCut$(H_2)$
 5.     if $|\delta(S_1)| \leq |\delta(S_2)|$ then
 6.         return $S_1$
 7.     else
 8.         return $S_2$
 9. else
10.     Compute an optimal solution $S$ by enumerating all solutions
11.     return $S$
```

Since the algorithm is recursive, determining its running time requires the analysis of a recurrence. Let $T(n)$ be the running time for a graph with n nodes. We know that there is a constant $c > 0$ such that `Contract(`G, t`)` needs at most $c \cdot n^2$ time. Thus, `FastCut(`G`)` spends at most $2c \cdot n^2$ time for the calls to `Contract(`G, t`)`. Aside from that, all steps need some constant $c' > 0$ time, except for the recursive calls. When $n < 7$, then the running time is bounded by some constant c'''. We get that the running time is bounded by the following recurrence:

$$T(n) \le \begin{cases} c''' & n < 7 \\ 2 \cdot T(\lceil n/\sqrt{2} \rceil) + 2cn^2 + c' & n \ge 7 \end{cases}$$

Solving this recurrence is a standard task that we leave as an exercise.

Lemma 1.16. *The running time of algorithm* `FastCut(G)` *is* $\mathcal{O}(n^2 \log n)$.

This is a much better bound than the running time bound we got for the `Contract` algorithm. Now the important part is to show that the `FastCut` algorithm has a good success probability.