1. Introduction
   1. Definition and examples of cut, min-unweighted cut
      1. Motivation by simple real-world example ← I haven’t found a suitable one, yet, so maybe we have to leave this out. The mail order company catalog problem from the PDF seems decent. Maybe we can singaporify that to make it more fun and catch people’s attention.
      2. A graph cut can be seen as a subset S of vertices. The weight of this cut is the total weight (or number) of edges with one endpoint in S and the other in V\S. Alternatively, a graph cut can be seen as a set of edges that, if removed from the graph, split it into two components.
      3. Make several examples (prepared, but maybe on the board) of graph cuts. Tell people that, in particular, one can just remove one node alone and its degree becomes the size of that cut.
      4. State the (weighted) min-cut problem (finding a graph cut with minimum weight). Also state related problems explicitly, to avoid confusion. So we say what we are \*not\* talking about (at least for the Karger-part): s-t-cuts, multiway-cuts. So: We are allowed to cut anywhere, the only requirement is that the graph actually falls apart into two components.
   2. Karger’s algorithm and example
      1. Do a little preview: We will explain an algorithm that runs in O(...) time and succeeds with … probability. So when we run it … times, it takes O(...) overall and we can be … sure to succeed. Later on, we will see an improvement of this algorithm that only takes O(...).
      2. Categorize the algorithm: Note the running time O(...) of a trivial algorithm that simply checks all possible cuts. That gives people an idea of how hard the problem actually might be and how good the algorithms are. Is it NP-complete? If so, we should mention.
      3. Then: Actually there are many ways to tackle the min-cut problem. For example, you can reduce it to LinProg (easily? if we pretend that it’s easy, we should prepare a backup slide to show how. That would certainly be interesting) or you can solve it via max-flow (we should look into how that’s done). The cool thing about Karger’s though is that it’s not based on any other technique. It’s just a plain min-cut algorithm.
      4. State Karger’s algorithm.
      5. Leave the (short) algorithm code visible at the top while going through an example underneath.
      6. When the algorithm is finished, show (backwards) how the final contracted graph relates to a cut in the original graph.
      7. Formally state the used notation of contraction (G/{v,w}).
      8. Explain how the algorithm reaches its running time. (this could be moved to part 2)
2. 4 facts and the probability of finding the minimum cut
   1. Fact 1 is straightforward.
   2. Fact 2 is straightforward, but must not be presented too quickly.
   3. Fact 3 needs proper explaining, maybe a hint to the pigeon hole principle.
   4. Fact 4 may be straight forward, but needs to be presented slowly. Especially this can be used as a bridge into the main analysis part: If the graph is large, the chance that we accidentally choose an edge that lies across a cut is pretty low.
   5. Explain how “contracting” an edge makes it impossible for this edge to occur in the final cut. So explain the relationship between cutting and contracting.
   6. Present the calculations for the probability that a certain (fixed) minimum cut is reached. This needs more detail than in the PDF. Every single “=” and “>=” should be crystal-clear, as this is the core of Karger’s insights.
   7. Starting off from the 1/{n choose 2} success probability, explain every single move in the calculation with running it l\*{n choose 2} times to reach 1-e^(-l) and how that connects to O(n^4) and 1/poly(n) for the chosen l.
3. (bridge to) improved version
   1. Go back to the 2/n failure formula derived in the previous part (currently 2.d.). For big graphs, making a mistake is less likely. But during the course of the algorithm, the graph shrinks. Thus, in the end, we’re more likely to make a mistake. To fix that mistake, we should try different alternatives and then choose the better one. But, as we saw before, trying different alternatives for the whole algorithm results in O(n^4). So instead, we should only retry the later parts of the algorithm, where the graph is already small and errors are more likely. So let’s say, we stop as soon as only k nodes are left (this is the “second” l from the slides but we should rename it to avoid confusion).
   2. If we stop at k nodes, the probability that a certain, fixed minimum cut is still there (i.e. not destroyed) is {k choose 2} / {n choose 2}. Optimally before showing this formula, we should explain it by deriving it from the telescope product. Essentially, you leave out the last k factors of that product, which results in the mentioned formula if you wite it cleverly.
   3. Then we just state that k=n/sqrt(2) is a clever choice and demonstrate why by plugging it in there (should give us >=0.5).
   4. State the recursive algorithm. Make sure that everyone understands how the recursion is happening. Especially that the later part of the graph is processed twice. Also make sure that people know the quantities here. One recursion is not one edge. It is (n-k) edges (all but the remaining k). And then, when we recursively call, the k becomes the new k. So the second time, we have k=N/2 where N is the original size, and then we have k=N/sqrt(2^3).
   5. Maybe d can be explained with an example. If full examples become too big (or unmeaningful if too small) we could design some graph sketches that we can draw on the board to better visualize what’s happening). Maybe we can have a graph (that the algo operates on) alongside a recursion tree (that visualizes how the algo operates).
   6. Quickly mention correctness (i.e. what we return is still a cut).
   7. Explain the runtime using the recursion formula and the master theorem (doesn’t need to be too detailed).
   8. Explain the success probability (in full detail).
   9. Finally have some nice table that compares the original, the improved algo and maybe the trivial brute-force one mentioned in the beginning.
4. variations / extensions
   1. Relationship to Kruskal’s algorithm?
   2. Beta(X)
   3. weighted
   4. Direct extension to multiway cut (with multiple islands).
5. Applications
   1. There seem to be a number of reasonably “close” applications. It’s important that the audience understands how the application is actually exactly the problem that we talked about earlier and that they are confident that the algorithms we presented could actually solve that.
   2. <http://labs.xjtudlc.com/labs/wldmt/reading%20list/books/Algorithms%20and%20optimization/Network%20Flows%20Theory,%20Algorithms,%20and%20Applications.pdf>
   3. The original papers by Karger are also good sources.
   4. In computer graphics there seem to be many applications. It would be hard to leave that out if talking about applications, but if we mention a concrete one, we should make sure that it’s well connected to min-cut and everyone sees on a low level how that is related.
   5. <http://riot.ieor.berkeley.edu/Applications/WeightedMinCut/>
   6. http://www.cs.princeton.edu/courses/archive/spr03/cs226/assignments/baseball.html