Approximation Algorithm for Maximum Cut

September 3, 2013

° rev. 2367ec4

Maximum Cut

Consider an undirected graph $G = (V, E)$ with positive edge weights $w(e)$ $(e \in E)$. Set $n = |V|$ and $m = |E|$. The Maximum Cut problem (Maxcut) is to find a partition of the vertices with the largest total weight of the edges "crossing" the partition, i.e., maximising the value of

$$c(A) = \sum_{(u,v)\in E, u\in A, v\notin A} w(u,v)$$

over all $A \subseteq V$.

Maxcut is NP-hard, so we have little hope of writing an algorithm that solves arbitrary Maxcut instances optimally.

Algorithm R

Consider the following simple randomized algorithm (call it Algorithm R): Let A be a *random subset* of V constructed by flipping a coin $r(v) \in \{0,1\}$ for every vertex $v \in V$ and setting $v \in A$ if and only if $r(v) = 1$.

Inputs

The data directory contains two input instances:

pw09_100.9.txt: A random instance with $|V| = 100$ and $|E| = 4455$. The best cut in this instance is 13658.[1]

matching_1000.txt: A disjoint union of 500 edges with unit weight.

[1] A. Wiegele, Biq Mac Library - A collection of Max-Cut and quadratic 0-1 programming instances of medium size, 2007.

The input format is straightforward: the first line contains n and m; every following line describes an edge in the format first vertex, second vertex, weight. All weights are integers, vertices are numbered $1, 2, \ldots, n$.

Deliverables

1. Implement algorithm R and run it on the dataset provided in the data directory. Use whatever programming language and libraries you want, but make sure that your code is short and crisp; my own solution takes 25 lines. Attach a printout to the report or have it checked by your lab assistant.
2. Fill out the report on the next page; you can just use the LaTeX code if you want.

Maxcut Lab Report

by Alice Cooper and Bob Marley[2]

[2] Complete the report by filling in your names and the parts marked [...]. Remove the sidenotes in your final hand-in.

Running time

The running time of algorithm R is [...][3].

[3] Replace [...] by a function of n and/or m. You can use asymptotic notation. This is supposed to be easy.

Randomness

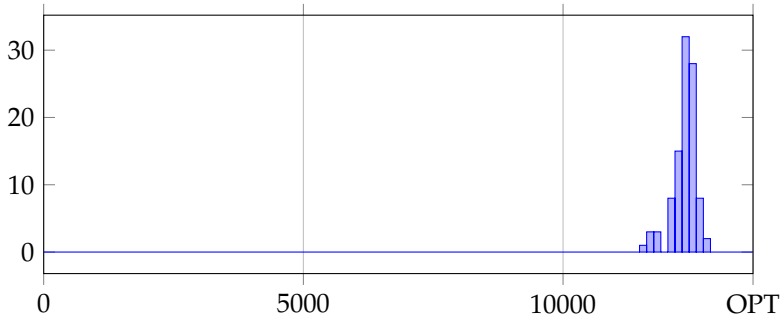
Algorithm R uses [...][4] random bits.

[4] Replace [...] by a function of n and/or m. Do not use asymptotic notation. This is supposed to be easy.

Solution quality

Experiments.

1. For the input file pw09_100.9.txt with $t = 100$ runs, we found an average cutsize of $C = [\ldots]$, roughly [...]% of the optimum OPT = 13658. The distribution of cutsizes looks as follows:[5]



[5] Display your cutsizes as a histogram. Use whatever software you like to produce the image; the placeholder image on the left is constructed in the LaTeX source.

2. For the input file matching_1000.txt [...][6]

[6] Perform the same analysis for matching_1000.txt. This involves thinking to determine OPT.

Analysis of performance guarantee Clearly, Algorithm R performs quite badly on input matching_1000.txt. We will show that it can perform *no worse* than that, i.e., we will establish that in expectation, the cutsize C satisfies $C \geq [\ldots] \cdot \mathrm{OPT}$.[7]

[7] Replace [...] by the right constant

We will view C as a random variable that gives the size of the cut defined by the random choices. Let W denote the total weight of the edges of G, i.e.,

$$W = \sum_{e \in E} w(e)\,.$$

Then,

$$E[C] = \tfrac{1}{2} \cdot W\,. \tag{1}$$

To see this, define the indicator random variable X_{uv} for every edge $uv \in E$ as follows. Set $X_{uv} = 1$ if uv crosses the cut, i.e., $u \in A$ and $v \notin A$ or $u \notin A$ and $v \in A$. Otherwise, $X_{uv} = 0$.

Then, $\Pr(X_{uv} = 1) = [\ldots]$. Now, $E[C] = [\ldots]$ Finally, we have $E[C] \geq [\ldots] \cdot \mathrm{OPT}$ because clearly [...].[8]

[8] Fill in the missing blanks in this paragraph. Your calculations and arguments need to include phrases like "because BLA and BLA are independent" or "disjoint," and "by linearity of expectation" and "because the weights are positive."

Optional: Derandomising Algorithm R

Algorithm L

We now reduce the number of random bits used by the algorithm to $\log n$ using a simple *pseudorandom generator*.

Let $k = \lceil \log(n+1) \rceil$ and flip k coins $b_1, \ldots, b_k$. There are $2^k - 1 \geq n$ different ways of choosing a nonempty subset $S \subseteq [k]$ of the coins. Each of these ways defines a random bit $r_S = \bigoplus_{i \in S} b_i$. (Here, $\oplus$ denotes exclusive or.) This gives a total of n random bits. These random bits are not as high-quality as the original k bits, but they retain the crucial property of *pairwise independence*: If $S \neq T$ then

$$\Pr(r_S \neq r_T) = [\ldots],.$$

Extend Algorithm R using this idea; call the resulting algorithm L (for logarithmic randomness).

Algorithm Z

For our final trick, we let the random bits disappear completely: since Algorithm L uses only k bits of randomness, we can iterate over *all* coin flips—there are only 2^k, which is polynomial (in fact, linear) in n. Extend algorithm L using this idea; call the resulting algorithm Z (for zero randomness). The running time of Z is $O([\ldots])$.

Perspective

This lab establishes minimal skills in algorithms implementation, probabilistic analysis of algorithms (independence, linearity of expectation, and in particular the trick of computing an expectation using indicator random variables), and approximation guarantees (in particular, finding upper and lower bounds by exhibiting a concrete "bad instance" and a comparison to a hypothetical optimum, respectively). The histogram aims to establish the intuition that measure is concentrated around its expectation.

To establish that Maxcut is NP-hard one reduces from NAE-Sat, a reduction that can be found in many places[9] Recall that the related problem *Minimum Cut* is easy because of the max flow–min cut theorem. A moment's thought should convince you that as soon as negative weights are allowed, the two problems are the same (and both are hard). Algorithm R doesn't work at all for negative weights.

[9] C. Moore and S. Mertens, *The Nature of Computation*, Oxford University Press, 2011, p. 146.

Algorithm R is a classical randomised approximation algorithm, its origins seem to be shrouded in the mists of time. The *deterministic* algorithm of Sahni and Gonzales[10] can be viewed as a derandomisation of R using the *method of conditional expectations*. These algorithms were best knows until the breakthrough result of Goemans and Williamson,[11] which improved the approximation factor to 0.87856. Håstad has shown that no algorithm can approximate the maxcut better than $16/17 \sim 0.941176$ unless P equals NP. Khot has shown that the Goemans–Williamson bound is essentially optimal under the *Unique Games Conjecture*.

[10] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. Assoc. Comput. Mach.*, 23(3):555–565, 1976.

[11] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.

Algorithm L can also be viewed as an application of *pairwise independent hash functions*.