Approximation Algorithm for Maximum Cut

September 12, 2017

[0] rev. 9a11012

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 $1, 2, \ldots, n$.

Algorithm S

Consider the following simple greedy swapping algorithm (call it Algorithm S): Let all the verticies be outside of A to begin with. A vetrex can be swapped, which means that if it's outside of A its moved into A and if its inside A its moved out of A. Pick the first vertex you can find that increases your cut if swapped. Swap this vetrex, and continue doing so until no vertex increases the cut if swapped, eg you find a local maxima.

Algorithm RS

Consider the following simple randomized swapping algorithm (call it Algorithm RS): Combine Algorithm R and S by instead of placing all vertices outside of A to begin with in S, place the verticies according to the output of R. Then proceed with the swapping part of S.

Deliverables

1. Implement algorithm R, S and RS 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; Each of the algorithms should not be much more than 20 lines. Reading input + scoring should also be small. Making all the code together less than 100 lines is a suitable goal. Attach a printout of the code to the report.
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

Algorithm R	Algorithm S	Algorithm RS
[...]	[...]	[...]

[3]

[3] Replace each [...] by a function of a subset of the parameters $\{n, m, W\}$, where $W = \sum_{e \in E} w(e)$. Use asymptotic notation.

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 for each algorithm, the average cutsize $Avg(C)$ and the maximum cutsize $Max(C)$:

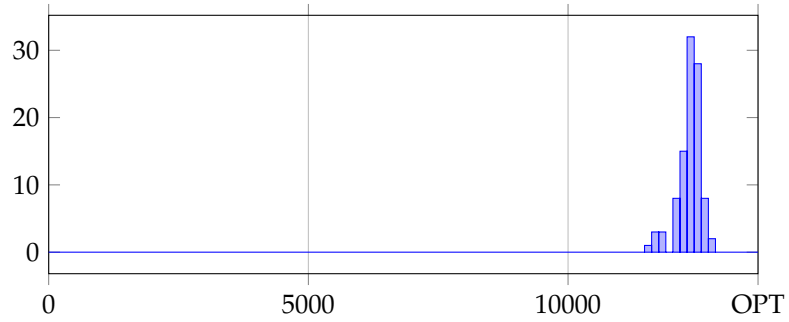
Algorithm	$Avg(C)$	$Max(C)$
R	[...]	[...]
S	[...]	[...]
RS	[...]	[...]

[5]

[5] Replace each [...] with the values obtained by running your experiments.

The optimum was given to us as OPT = 13658.

The distribution of cutsizes for Algorithm R looks as follows:[6]



[6] 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.

The distribution of cutsizes for Algorithm RS looks as follows: [...] [7]

[7] Display a plot similar to the histogram for Algorithm R, but instead for Algorithm RS.

2. For the input file matching_1000.txt [...][8]

[8] 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 $C \geq [\ldots] \cdot \mathrm{OPT}$.[9]

[9] 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, $X_{uv} = 0$.

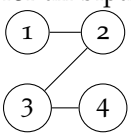
Then, $\Pr(X_{uv} = 1) = [\ldots]$. Now, $E[C] = [\ldots]$ Finally, we have $E[C] \geq [\ldots] \cdot \mathrm{OPT}$ because clearly $[\ldots]$.[10]

[10] 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."

Algorithms S and RS perform very well on input matching_1000.txt. In fact both algorithms always find the optimum. The reason for this is that $[\ldots]$ [11].

[11] Explain why the optimum is always found.

However Algorithm RS does not always find the optimum even for all bipartite graphs. An example of this scenario is the graph G,



Where the weight of the edges are

u	v	weight
1	2	$[\ldots]$
2	3	$[\ldots]$
3	4	$[\ldots]$

[12]

[12] Fill in weights of the edges in the graph such that the graph has a local maxima less than the global maxima.

Here if nodes $[\ldots]$ are inside A, Algorithm RS gets stuck in a local maxima of size $[\ldots]$. The global maxima has size $[\ldots]$, with nodes 1 and 3 inside A.[13]

[13] Fill in the blanks so the sentence makes sense.

Perspective

To establish that Maxcut is NP-hard one reduces from NAE-Sat, a reduction that can be found in many places[14] 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.

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[15] 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,[16] 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*.

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

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

[16] 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.