高级算法 (Fall 2019)/Min-Cut and Max-Cut

Contents

Graph Cut

Let $G(V,E)$ be an undirected graph. A subset $C \subseteq E$ of edges is a **cut** of graph G if G becomes *disconnected* after deleting all edges in C.

Let $\{S,T\}$ be a **bipartition** of V into nonempty subsets $S,T \subseteq V$, where $S \cap T = \emptyset$ and $S \cup T = V$. A cut C is specified by this bipartition as

$$C = E(S,T)\ ,$$

where $E(S,T)$ denotes the set of "crossing edges" with one endpoint in each of S and T, formally defined as

$$E(S,T) = \{uv \in E \mid u \in S, v \in T\}\ .$$

Given a graph G, there might be many cuts in G, and we are interested in finding the **minimum** or **maximum** cut.

Min-Cut

The **min-cut problem**, also called the **global minimum cut problem**, is defined as follows.

Min-cut problem

- **Input**: an undirected graph $G(V, E)$;
- **Output**: a cut C in G with the smallest size $|C|$.

Equivalently, the problem asks to find a bipartition of V into disjoint non-empty subsets S and T that minimizes $|E(S, T)|$.

We consider the problem in a slightly more generalized setting, where the input graphs G can be **multi-graphs**, meaning that there could be multiple **parallel edges** between two vertices u and v. The cuts in multi-graphs are defined in the same way as before, and the cost of a cut C is given by the total number of edges (including parallel edges) in C. Equivalently, one may think of a multi-graph as a graph with integer edge weights, and the cost of a cut C is the total weights of all edges in C.

A canonical deterministic algorithm for this problem is through the max-flow min-cut theorem (http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem). The max-flow algorithm finds us a minimum s-t **cut**, which disconnects a **source** $s \in V$ from a **sink** $t \in V$, both specified as part of the input. A global min cut can be found by exhaustively finding the minimum s-t cut for an arbitrarily fixed source s and all possible sink $t \neq s$. This takes $(n-1)\times$max-flow time where $n = |V|$ is the number of vertices.

The fastest known deterministic algorithm for the minimum cut problem on multi-graphs is the Stoer—Wagner algorithm (https://en.wikipedia.org/wiki/Stoer—Wagner_algorithm), which achieves an $O(mn + n^2 \log n)$ time complexity where $m = |E|$ is the total number of edges (counting the parallel edges).

If we restrict the input to be **simple graphs** (meaning there is no parallel edges) with no edge weight, there are better algorithms. A deterministic algorithm of Ken-ichi Kawarabayashi and Mikkel Thorup (https://dl.acm.org/citation.cfm?id=2746588) published in STOC 2015, achieves the near-linear (in the number of edges) time complexity.

Karger's *Contraction* algorithm

We will describe a simple and elegant randomized algorithm for the min-cut problem. The algorithm is due to David Karger (http://people.csail.mit.edu/karger/).

Let $G(V, E)$ be a **multi-graph**, which allows more than one **parallel edges** between two distinct vertices u and v but does not allow any **self-loops**: the edges that adjoin a vertex to itself. A multi-graph G can be represented by an adjacency matrix A, in the way that each non-diagonal entry $A(u, v)$ takes nonnegative integer values instead of just 0 or 1, representing the number of parallel edges between u and v in G, and all diagonal entries $A(v, v) = 0$ (since there is no self-loop).

Given a multi-graph $G(V, E)$ and an edge $e \in E$, we define the following **contraction** operator Contract(G, e), which transform G to a new multi-graph.

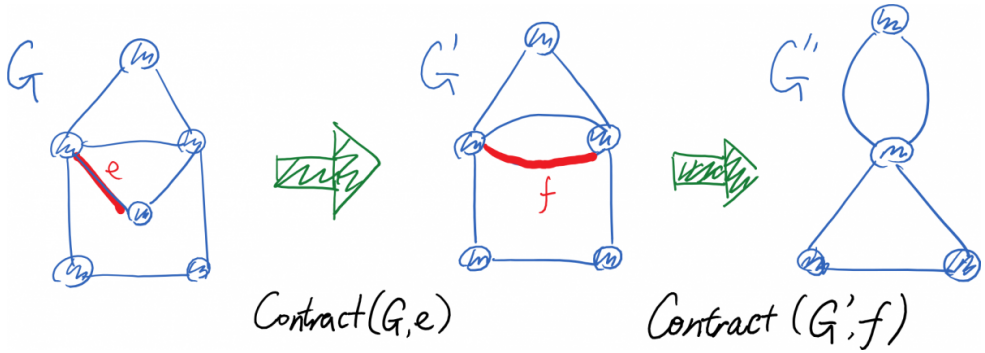
The contraction operator *Contract*(G, e)

say $e = uv$:

- replace $\{u, v\}$ by a new vertex x;
- for every edge (no matter parallel or not) in the form of uw or vw that connects one of $\{u, v\}$ to a vertex $w \in V \setminus \{u, v\}$ in the graph other than u, v, replace it by a new edge xw;
- the reset of the graph does not change.

In other words, the $Contract(G, uv)$ merges the two vertices u and v into a new vertex x whose incident edges preserves the edges incident to u or v in the original graph G except for the parallel edges between them. Now you should realize why we consider multi-graphs instead of simple graphs, because even if we start with a simple graph without parallel edges, the contraction operator may create parallel edges.

The contraction operator is illustrated by the following picture:



Karger's algorithm uses a simple idea:

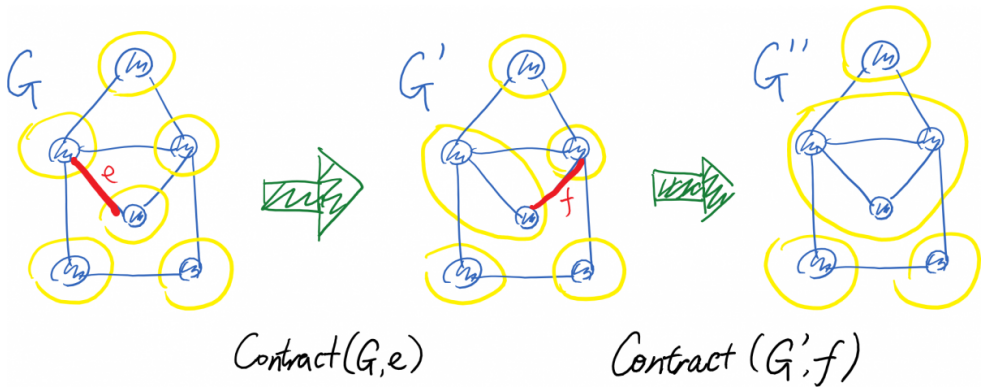
- At each step we randomly select an edge in the current multi-graph to contract until there are only two vertices left.
- The parallel edges between these two remaining vertices must be a cut of the original graph.
- We return this cut and hope that with good chance this gives us a minimum cut.

The following is the pseudocode for Karger's algorithm.

RandomContract **(Karger 1993)**
Input: multi-graph $G(V, E)$;
while $\|V\| > 2$ do ▪ choose an edge $uv \in E$ uniformly at random; ▪ $G = Contract(G, uv)$; return $C = E$ (the parallel edges between the only two vertices in V);

Another way of looking at the contraction operator Contract(G,e) is that we are dealing with classes of vertices. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of all vertices. We start with n vertex classes $S_1, S_2, \ldots, S_n$ with each class $S_i = \{v_i\}$ contains one vertex. By calling $Contract(G, uv)$, where $u \in S_i$ and $v \in S_j$ for distinct $i \neq j$, we take union of S_i and S_j. The edges in the contracted multi-graph are the edges that cross between different vertex classes.

This view of contraction is illustrated by the following picture:



The following claim is left as an exercise for the class:

- With suitable choice of data structures, each operation $Contract(G, e)$ can be implemented within running time $O(n)$ where $n = |V|$ is the number of vertices.

In the above ***RandomContract*** algorithm, there are precisely $n - 2$ contractions. Therefore, we have the following time upper bound.

Theorem

For any multigraph with n vertices, the running time of the ***RandomContract*** algorithm is $O(n^2)$.

We emphasize that it's the time complexity of a "single running" of the algorithm: later we will see we may need to run this algorithm for many times to guarantee a desirable accuracy.

Analysis of accuracy

We now analyze the performance of the above algorithm. Since the algorithm is ***randomized***, its output cut is a random variable even when the input is fixed, so *the output may not always be correct*. We want to give a theoretical guarantee of the chance that the algorithm returns a correct answer on an arbitrary input.

More precisely, on an arbitrarily fixed input multi-graph G, we want to answer the following question rigorously:

$$p_{\text{correct}} = \Pr[\text{ a minimum cut is returned by } RandomContract\ ] \geq ?$$

To answer this question, we prove a stronger statement: for arbitrarily fixed input multi-graph G and a particular minimum cut C in G,

$$p_C = \Pr[\ C \text{ is returned by } RandomContract\ ] \geq ?$$

Obviously this will imply the previous lower bound for p_{correct} because the event in p_C implies the event in p_{correct}.

- In above argument we use the simple law in probability that $\Pr[A] \leq \Pr[B]$ if $A \subseteq B$, i.e. event A implies event B.

We introduce the following notations:

- Let $e_1, e_2, \ldots, e_{n-2}$ denote the sequence of random edges chosen to contract in a running of *RandomContract* algorithm.
- Let $G_1 = G$ denote the original input multi-graph. And for $i = 1, 2, \ldots, n-2$, let $G_{i+1} = Contract(G_i, e_i)$ be the multigraph after ith contraction.

Obviously $e_1, e_2, \ldots, e_{n-2}$ are random variables, and they are the *only* random choices used in the algorithm: meaning that they along with the input G, uniquely determine the sequence of multi-graphs $G_1, G_2, \ldots, G_{n-2}$ in every iteration as well as the final output.

We now compute the probability p_C by decompose it into more elementary events involving $e_1, e_2, \ldots, e_{n-2}$. This is due to the following proposition.

Proposition 1

If C is a minimum cut in a multi-graph G and $e \notin C$, then C is still a minimum cut in the contracted graph $G' = contract(G, e)$.

Proof.

We first observe that contraction will never create new cuts: every cut in the contracted graph G' must also be a cut in the original graph G.

We then observe that a cut C in G "survives" in the contracted graph G' if and only if the contracted edge $e \notin C$.

Both observations are easy to verify by the definition of contraction operator (in particular, easier to verify if we take the vertex class interpretation). The detailed proofs are left as an exercise.

□

Recall that $e_1, e_2, \ldots, e_{n-2}$ denote the sequence of random edges chosen to contract in a running of *RandomContract* algorithm.

By Proposition 1, the event ``$C$ is returned by $RandomContract$'' is equivalent to the event ``$e_i \notin C$ for all $i = 1, 2, \ldots, n-2$'' . Therefore:

$$\begin{aligned} p_C &= \Pr[\, C \text{ is returned by } RandomContract \,] \\ &= \Pr[\, e_i \notin C \text{ for all } i = 1, 2, \ldots, n-2 \,] \\ &= \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid \forall j < i, e_j \notin C]. \end{aligned}$$

The last equation is due to the so called **chain rule** in probability.

- The **chain rule**, also known as the **law of progressive conditioning**, is the following proposition: for a sequence of events (not necessarily independent) $A_1, A_2, \ldots, A_n$,

  $$\Pr[\forall i, A_i] = \prod_{i=1}^{n} \Pr[A_i \mid \forall j < i, A_j] \ .$$

  It is a simple consequence of the definition of conditional probability. By definition of conditional probability,

  $$\Pr[A_n \mid \forall j < n] = \frac{\Pr[\forall i, A_i]}{\Pr[\forall j < n, A_j]} \ ,$$

  and equivalently we have

  $$\Pr[\forall i, A_i] = \Pr[\forall j < n, A_j] \Pr[A_n \mid \forall j < n] \ .$$

  Recursively apply this to $\Pr[\forall j < n, A_j]$ we obtain the chain rule.

Back to the analysis of probability p_C.

Now our task is to give lower bound to each $p_i = \Pr[e_i \notin C \mid \forall j < i, e_j \notin C]$. The condition ``$\forall j < i, e_j \notin C$'' means the min-cut C survives all first $i-1$ contractions $e_1, e_2, \ldots, e_{i-1}$, which due to Proposition 1 means that C is also a min-cut in the multi-graph G_i obtained from applying the first $(i-1)$ contractions.

Then the conditional probability $p_i = \Pr[e_i \notin C \mid \forall j < i, e_j \notin C]$ is the probability that no edge in C is hit when a uniform random edge in the current multi-graph is chosen assuming that C is a minimum cut in the current multi-graph. Intuitively this probability should be bounded from below, because as a min-cut C should be sparse among all edges. This intuition is justified by the following proposition.

Proposition 2

If C is a min-cut in a multi-graph $G(V, E)$, then $|E| \geq \frac{|V||C|}{2}$.

Proof.

It must hold that the degree of each vertex $v \in V$ is at least $|C|$, or otherwise the set of edges incident to v forms a cut of size smaller than $|C|$ which separates $\{v\}$ from the rest of the graph, contradicting that C is a min-cut. And the bound $|E| \geq \frac{|V||C|}{2}$ follows directly from applying the handshaking lemma (https://en.wikipedia.org/wiki/Handshaking_lemma) to the fact that every vertex in G has degree at least $|C|$.

□

Let V_i and E_i denote the vertex set and edge set of the multi-graph G_i respectively, and recall that G_i is the multi-graph obtained from applying first $(i-1)$ contractions. Obviously $|V_i| = n - i + 1$. And due to Proposition 2, $|E_i| \geq \frac{|V_i||C|}{2}$ if C is still a min-cut in G_i.

The probability $p_i = \Pr[e_i \notin C \mid \forall j < i, e_j \notin C]$ can be computed as

$$\begin{aligned} p_i &= 1 - \frac{|C|}{|E_i|} \\ &\geq 1 - \frac{2}{|V_i|} \\ &= 1 - \frac{2}{n-i+1} \end{aligned},$$

where the inequality is due to Proposition 2.

We now can put everything together. We arbitrarily fix the input multi-graph G and any particular minimum cut C in G.

$$\begin{aligned} p_{\text{correct}} &= \Pr[\text{ a minimum cut is returned by } \mathit{RandomContract}\,] \\ &\geq \Pr[\, C \text{ is returned by } \mathit{RandomContract}\,] \\ &= \Pr[\, e_i \notin C \text{ for all } i = 1, 2, \ldots, n-2\,] \\ &= \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid \forall j < i, e_j \notin C] \\ &\geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) \\ &= \prod_{k=3}^{n} \frac{k-2}{k} \\ &= \frac{2}{n(n-1)}. \end{aligned}$$

This gives us the following theorem.

Theorem

For any multigraph with n vertices, the *RandomContract* algorithm returns a minimum cut with probability at least $\frac{2}{n(n-1)}$.

At first glance this seems to be a miserable chance of success. However, notice that there may be exponential many cuts in a graph (because potentially every nonempty subset $S \subset V$ corresponds to a cut $C = E(S, \overline{S})$), and Karger's algorithm effectively reduce this exponential-sized space of feasible solutions to a quadratic size one, an exponential improvement!

We can run *RandomContract* independently for $t = \frac{n(n-1)\ln n}{2}$ times and return the smallest cut ever returned. The probability that a minimum cut is found is at least:

$$\begin{aligned}
&1 - \Pr[\text{ all } t \text{ independent runnings of } \mathit{RandomContract} \text{ fails to find a min-cut }] \\
&= 1 - \Pr[\text{ a single running of } \mathit{RandomContract} \text{ fails }]^t \\
&\geq 1 - \left(1 - \frac{2}{n(n-1)}\right)^{\frac{n(n-1)\ln n}{2}} \\
&\geq 1 - \frac{1}{n}.
\end{aligned}$$

Recall that a running of *RandomContract* algorithm takes $O(n^2)$ time. Altogether this gives us a randomized algorithm running in time $O(n^4 \log n)$ and find a minimum cut **with high probability** (https://en.wikipedia.org/wiki/With_high_probability).

A Corollary by the Probabilistic Method

The analysis of Karger's algorithm implies the following combinatorial proposition for the number of distinct minimum cuts in a graph.

Corollary

For any graph $G(V, E)$ of n vertices, the number of distinct minimum cuts in G is at most $\frac{n(n-1)}{2}$.

Proof.

Let $\mathcal{C}$ denote the set of all minimum cuts in G. For each min-cut $C \in \mathcal{C}$, let A_C denote the event "C is returned by *RandomContract*", whose probability is given by

$$p_C = \Pr[A_C] .$$

Clearly we have:

- for any distinct $C, D \in \mathcal{C}$, A_C and A_D are **disjoint events**; and
- the union $\bigcup_{C \in \mathcal{C}} A_C$ is precisely the event "a minimum cut is returned by *RandomContract*", whose probability is given by

$$p_{\text{correct}} = \Pr[\text{ a minimum cut is returned by } \mathit{RandomContract}] .$$

Due to the **additivity of probability** (https://en.wikipedia.org/wiki/Probability_axioms#Third_axiom), it holds that

$$p_{\text{correct}} = \sum_{C \in \mathcal{C}} \Pr[A_C] = \sum_{C \in \mathcal{C}} p_C .$$

By the analysis of Karger's algorithm, we know $p_C \geq \frac{2}{n(n-1)}$. And since p_{correct} is a well defined probability, due to the **unitarity of probability** (https://en.wikipedia.org/wiki/Probability_axioms#Second_axiom), it must hold that $p_{\text{correct}} \leq 1$. Therefore,

$$1 \geq p_{\text{correct}} = \sum_{C \in \mathcal{C}} p_C \geq |\mathcal{C}| \frac{2}{n(n-1)} ,$$

which means $|\mathcal{C}| \leq \frac{n(n-1)}{2}$.

□

Note that the statement of this theorem has no randomness at all, while the proof consists of a randomized procedure. This is an example of the probabilistic method (http://en.wikipedia.org/wiki/Probabilistic_method).

Fast Min-Cut

In the analysis of *RandomContract* algorithm, recall that we lower bound the probability p_C that a min-cut C is returned by *RandomContract* by the following **telescopic product**:

$$p_C \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) .$$

Here the index i corresponds to the ith contraction. The factor $\left(1 - \frac{2}{n-i+1}\right)$ is decreasing in i, which means:

- The probability of success is only getting bad when the graph is getting "too contracted", that is, when the number of remaining vertices is getting small.

This motivates us to consider the following alternation to the algorithm: first using random contractions to reduce the number of vertices to a moderately small number, and then recursively finding a min-cut in this smaller instance. This seems just a restatement of exactly what we have been doing. Inspired by the idea of boosting the accuracy via independent repetition, here we apply the recursion on *two* smaller instances generated independently.

The algorithm obtained in this way is called *FastCut*. We first define a procedure to randomly contract edges until there are t number of vertices left.

RandomContract(G, t)

Input: multi-graph $G(V, E)$, and integer $t \geq 2$;

---

while $|V| > t$ do

- choose an edge $uv \in E$ uniformly at random;
- $G = Contract(G, uv)$;

return G;

The *FastCut* algorithm is recursively defined as follows.

FastCut(G)

Input: multi-graph $G(V, E)$;

if $|V| \leq 6$ then return a mincut by brute force;
else let $t = \lceil 1 + |V|/\sqrt{2} \rceil$;

$G_1 = RandomContract(G, t)$;
$G_2 = RandomContract(G, t)$;
return the smaller one of $FastCut(G_1)$ and $FastCut(G_2)$;

As before, all G are multigraphs.

Fix a min-cut C in the original multigraph G. By the same analysis as in the case of *RandomContract*, we have

$$\begin{aligned}
&\Pr[C \text{ survives all contractions in } RandomContract(G,t)] \\
=&\prod_{i=1}^{n-t} \Pr[C \text{ survives the } i\text{-th contraction} \mid C \text{ survives the first } (i-1)\text{-th contractions}] \\
\geq&\prod_{i=1}^{n-t} \left(1 - \frac{2}{n-i+1}\right) \\
=&\prod_{k=t+1}^{n} \frac{k-2}{k} \\
=&\frac{t(t-1)}{n(n-1)}.
\end{aligned}$$

When $t = \lceil 1 + n/\sqrt{2} \rceil$, this probability is at least $1/2$. The choice of t is due to our purpose to make this probability at least $1/2$. You will see this is crucial in the following analysis of accuracy.

We denote by A and B the following events:

A : C survives all contractions in $RandomContract(G,t)$;

B : size of min-cut is unchanged after $RandomContract(G,t)$;

Clearly, A implies B and by above analysis $\Pr[B] \geq \Pr[A] \geq \frac{1}{2}$.

We denote by $p(n)$ the lower bound on the probability that $FastCut(G)$ succeeds for a multigraph of n vertices, that is

$$p(n) = \min_{G:|V|=n} \Pr[\ FastCut(G) \text{ returns a min-cut in } G\ ].$$

Suppose that G is the multigraph that achieves the minimum in above definition. The following recurrence holds for $p(n)$.

$$\begin{aligned}
p(n) &= \Pr[\ FastCut(G) \text{ returns a min-cut in } G\ ] \\
&= \Pr[\ \text{ a min-cut of } G \text{ is returned by } FastCut(G_1) \text{ or } FastCut(G_2)\ ] \\
&\geq 1 - (1 - \Pr[B \wedge FastCut(G_1) \text{ returns a min-cut in } G_1\ ])^2 \\
&\geq 1 - (1 - \Pr[A \wedge FastCut(G_1) \text{ returns a min-cut in } G_1\ ])^2 \\
&= 1 - (1 - \Pr[A]\Pr[FastCut(G_1) \text{ returns a min-cut in } G_1 \mid A])^2 \\
&\geq 1 - \left(1 - \frac{1}{2} p\left(\left\lceil 1 + n/\sqrt{2} \right\rceil\right)\right)^2,
\end{aligned}$$

where A and B are defined as above such that $\Pr[A] \geq \frac{1}{2}$.

The base case is that $p(n) = 1$ for $n \leq 6$. By induction it is easy to prove that

$$p(n) = \Omega\left(\frac{1}{\log n}\right).$$

Recall that we can implement an edge contraction in $O(n)$ time, thus it is easy to verify the following recursion of time complexity:

$$T(n) = 2T\left(\left\lceil 1 + n/\sqrt{2} \right\rceil\right) + O(n^2),$$

where $T(n)$ denotes the running time of $FastCut(G)$ on a multigraph G of n vertices.

By induction with the base case $T(n) = O(1)$ for $n \leq 6$, it is easy to verify that $T(n) = O(n^2 \log n)$.

Theorem

For any multigraph with n vertices, the *FastCut* algorithm returns a minimum cut with probability $\Omega\left(\frac{1}{\log n}\right)$ in time $O(n^2 \log n)$.

At this point, we see the name *FastCut* is misleading because it is actually slower than the original *RandomContract* algorithm, only the chance of successfully finding a min-cut is much better (improved from an $\Omega(1/n^2)$ to an $\Omega(1/\log n)$).

Given any input multi-graph, repeatedly running the *FastCut* algorithm independently for some $O((\log n)^2)$ times and returns the smallest cut ever returned, we have an algorithm which runs in time $O(n^2 \log^3 n)$ and returns a min-cut with probability $1 - O(1/n)$, i.e. with high probability.

Recall that the running time of best known deterministic algorithm for min-cut on multi-graph is $O(mn + n^2 \log n)$. On dense graph, the randomized algorithm outperforms the best known deterministic algorithm.

Finally, Karger further improves this and obtains a near-linear (in the number of edges) time randomized algorithm (https://arxiv.org/abs/cs/9812007) for minimum cut in multi-graphs.

Max-Cut

The **maximum cut problem**, in short the **max-cut problem**, is defined as follows.

Max-cut problem
■ **Input**: an undirected graph $G(V, E)$; ■ **Output**: a bipartition of V into disjoint subsets S and T that maximizes $\lvert E(S, T)\rvert$.

The problem is a typical MAX-CSP, an optimization version of the constraint satisfaction problem (https://en.wikipedia.org/wiki/Constraint_satisfaction_problem). An instance of CSP consists of:

- a set of variables $x_1, x_2, \ldots, x_n$ usually taking values from some finite domain;
- a sequence of constraints (predicates) $C_1, C_2, \ldots, C_m$ defined on those variables.

The MAX-CSP asks to find an assignment of values to variables $x_1, x_2, \ldots, x_n$ which maximizes the number of satisfied constraints.

In particular, when the variables $x_1, x_2, \ldots, x_n$ takes Boolean values $\{0, 1\}$ and every constraint is a binary constraint $\cdot \neq \cdot$ in the form of $x_1 \neq x_j$, then the MAX-CSP is precisely the max-cut problem.

Unlike the min-cut problem, which can be solved in polynomial time, the max-cut is known to be **NP-hard** (https://en.wikipedia.org/wiki/NP-hardness). Its decision version is among the 21 **NP-complete** problems found by Karp (https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems). This means we should not hope for a polynomial-time algorithm for solving the problem if a famous conjecture in computational complexity (https://en.wikipedia.org/wiki/P_versus_NP_problem) is correct. And due to another less famous conjecture in computational complexity (https://en.wikipedia.org/wiki/BPP_(complexity)#Problems), randomization alone probably cannot help this situation either.

We may compromise our goal and allow algorithm to *not always find the optimal solution*. However, we still want to guarantee that the algorithm *always returns a relatively good solution on all possible instances*. This notion is formally captured by **approximation algorithms** and **approximation ratio**.

Greedy algorithm

A natural heuristics for solving the max–cut is to sequentially join the vertices to one of the two disjoint subsets S and T to *greedily* maximize the *current* number of edges crossing between S and T.

To state the algorithm, we overload the definition $E(S,T)$. Given an undirected graph $G(V,E)$, for any disjoint subsets $S,T \subseteq V$ of vertices, we define

$$E(S,T) = \{uv \in E \mid u \in S, v \in T\} \ .$$

We also assume that the vertices are ordered arbitrarily as $V = \{v_1, v_2, \ldots, v_n\}$.

The greedy heuristics is then described as follows.

GreedyMaxCut
Input: undirected graph $G(V,E)$, with an arbitrary order of vertices $V = \{v_1, v_2, \ldots, v_n\}$;
initially $S = T = \emptyset$; for $i = 1, 2, \ldots, n$ v_i joins one of S, T to maximize the current $\lvert E(S,T)\rvert$ (breaking ties arbitrarily);

The algorithm certainly runs in polynomial time.

Without any guarantee of how good the solution returned by the algorithm approximates the optimal solution, the algorithm is only a heuristics, not an **approximation algorithm**.

Approximation ratio

For now we restrict ourselves to the max–cut problem, although the notion applies more generally.

Let G be an arbitrary instance of max–cut problem. Let OPT_G denote the size of the of max–cut in graph G. More precisely,

$$OPT_G = \max_{S \subseteq V} |E(S, \overline{S})| \ .$$

Let SOL_G be the size of of the cut $\lvert E(S,T)\rvert$ returned by the *GreedyMaxCut* algorithm on input graph G.

As a maximization problem it is trivial that $SOL_G \leq OPT_G$ for all G. To guarantee that the *GreedyMaxCut* gives good approximation of optimal solution, we need the other direction:

Approximation ratio

We say that the **approximation ratio** of the *GreedyMaxCut* algorithm is α, or *GreedyMaxCut* is an α**−approximation** algorithm, for some $0 < \alpha \leq 1$, if

$\frac{SOL_G}{OPT_G} \geq \alpha$ for every possible instance G of max−cut.

With this notion, we now try to analyze the approximation ratio of the *GreedyMaxCut* algorithm.

A dilemma to apply this notion in our analysis is that in the definition of approximation ratio, we compare the solution returned by the algorithm with the **optimal solution**. However, in the analysis we can hardly conduct similar comparisons to the optimal solutions. A fallacy in this logic is that the optimal solutions are **NP−hard**, meaning there is no easy way to calculate them (e.g. a closed form).

A popular step (usually the first step of analyzing approximation ratio) to avoid this dilemma is that instead of directly comparing to the optimal solution, we compare to an **upper bound** of the optimal solution (for minimization problem, this needs to be a lower bound), that is, we compare to something which is even better than the optimal solution (which means it cannot be realized by any feasible solution).

For the max−cut problem, a simple upper bound to OPT_G is $|E|$, the number of all edges. This is a trivial upper bound of max−cut since any cut is a subset of edges.

Let $G(V, E)$ be the input graph and $V = \{v_1, v_2, \ldots, v_n\}$. Initially $S_1 = T_1 = \emptyset$. And for $i = 1, 2, \ldots, n$, we let S_{i+1} and T_{i+1} be the respective S and T after v_i joins one of S, T. More precisely,

- $S_{i+1} = S_i \cup \{v_i\}$ and $T_{i+1} = T_i$ if $E(S_i \cup \{v_i\}, T_i) > E(S_i, T_i \cup \{v_i\})$;
- $S_{i+1} = S_i$ and $T_{i+1} = T_i \cup \{v_i\}$ if otherwise.

Finally, the max−cut is given by

$$SOL_G = |E(S_{n+1}, T_{n+1})|.$$

We first observe that we can count the number of edges $|E|$ by summarizing the contributions of individual v_i's.

Proposition 1

$$|E| = \sum_{i=1}^{n} \left(|E(S_i, \{v_i\})| + |E(T_i, \{v_i\})|\right).$$

Proof.

Note that $S_i \cup T_i = \{v_1, v_2, \ldots, v_{i-1}\}$, i.e. S_i and T_i together contain precisely those vertices preceding v_i. Therefore, by taking the sum

$$\sum_{i=1}^{n} \left(|E(S_i, \{v_i\})| + |E(T_i, \{v_i\})|\right),$$

we effectively enumerate all (v_j, v_i) that $v_j v_i \in E$ and $j < i$. The total number is precisely $|E|$.

□

We then observe that the SOL_G can be decomposed into contributions of individual v_i's in the same way.

Proposition 2

$SOL_G = \sum_{i=1}^{n} \max\left(|E(S_i, \{v_i\})|, |E(T_i, \{v_i\})|\right)$.

Proof.

It is east to observe that $E(S_i, T_i) \subseteq E(S_{i+1}, T_{i+1})$, i.e. once an edge joins the cut between current S, T it will never drop from the cut in the future.

We then define

$$\Delta_i = |E(S_{i+1}, T_{i+1})| - |E(S_i, T_i)| = |E(S_{i+1}, T_{i+1}) \setminus E(S_i, T_i)|$$

to be the contribution of v_i in the final cut.

It holds that

$$\sum_{i=1}^{n} \Delta_i = |E(S_{n+1}, T_{n+1})| - |E(S_1, T_1)| = |E(S_{n+1}, T_{n+1})| = SOL_G \ .$$

On the other hand, due to the greedy rule:

- $S_{i+1} = S_i \cup \{v_i\}$ and $T_{i+1} = T_i$ if $E(S_i \cup \{v_i\}, T_i) > E(S_i, T_i \cup \{v_i\})$;
- $S_{i+1} = S_i$ and $T_{i+1} = T_i \cup \{v_i\}$ if otherwise;

it holds that

$$\Delta_i = |E(S_{i+1}, T_{i+1}) \setminus E(S_i, T_i)| = \max\left(|E(S_i, \{v_i\})|, |E(T_i, \{v_i\})|\right) \ .$$

Together the proposition follows.

□

Combining the above Proposition 1 and Proposition 2, we have

$$\begin{aligned} SOL_G &= \sum_{i=1}^{n} \max\left(|E(S_i, \{v_i\})|, |E(T_i, \{v_i\})|\right) \\ &\geq \frac{1}{2} \sum_{i=1}^{n} \left(|E(S_i, \{v_i\})| + |E(T_i, \{v_i\})|\right) \\ &= \frac{1}{2}|E| \\ &\geq \frac{1}{2} OPT_G. \end{aligned}$$

Theorem

The *GreedyMaxCut* is a 0.5−approximation algorithm for the max−cut problem.

This is not the best approximation ratio achieved by polynomial−time algorithms for max−cut.

- The best known approximation ratio achieved by any polynomial−time algorithm is achieved by the Goemans−Williamson algorithm (http://www−math.mit.edu/~goemans/PAPERS/maxcut−jacm.pdf), which relies on rounding an SDP (https://en.wikipedia.org/wiki/Semidefinite_programming) relaxation of the max−cut, and achieves an approximation ratio $\alpha^* \approx 0.878$, where α^* is an irrational whose precise value is given by $\alpha^* = \frac{2}{\pi} \inf_{x \in [-1,1]} \frac{\arccos(x)}{1-x}$.
- Assuming the unique game conjecture (https://en.wikipedia.org/wiki/Unique_games_conjecture), there does not exist any polynomial−time algorithm for max−cut with approximation ratio $\alpha > \alpha^*$.

Derandomization by conditional expectation

There is a probabilistic interpretation of the greedy algorithm, which may explains why we use greedy scheme for max−cut and why it works for finding an approximate max−cut.

Given an undirected graph $G(V, E)$, let us calculate the average size of cuts in G. For every vertex $v \in V$ let $X_v \in \{0, 1\}$ be a *uniform* and *independent* random bit which indicates whether v joins S or T. This gives us a uniform random bipartition of V into S and T.

The size of the random cut $|E(S, T)|$ is given by

$$|E(S,T)| = \sum_{uv \in E} I[X_u \neq X_v],$$

where $I[X_u \neq X_v]$ is the Boolean indicator random variable that indicates whether event $X_u \neq X_v$ occurs.

Due to **linearity of expectation**,

$$\mathbb{E}[|E(S,T)|] = \sum_{uv \in E} \mathbb{E}[I[X_u \neq X_v]] = \sum_{uv \in E} \Pr[X_u \neq X_v] = \frac{|E|}{2}.$$

Recall that $|E|$ is a trivial upper bound for the max-cut OPT_G. Due to the above argument, we have

$$\mathbb{E}[|E(S,T)|] \geq \frac{OPT_G}{2}.$$

- In above argument we use a few probability propositions.

  linearity of expectation:

  Let $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ be a random vector. Then

  $$\mathbb{E}\left[\sum_{i=1}^{n} c_i X_i\right] = \sum_{i=1}^{n} c_i \mathbb{E}[X_i],$$

  where $c_1, c_2, \ldots, c_n$ are scalars.
  That is, the order of computations of expectation and linear (affine) function of a random vector can be exchanged.
  Note that this property ignores the dependency between random variables, and hence is very useful.

  Expectation of indicator random variable:

  We usually use the notation $I[A]$ to represent the Boolean indicator random variable that indicates whether the event A occurs: i.e. $I[A] = 1$ if event A occurs and $I[A] = 0$ if otherwise.
  It is easy to see that $\mathbb{E}[I[A]] = \Pr[A]$. The expectation of an indicator random variable equals the probability of the event it indicates.

By above analysis, the average (under uniform distribution) size of all cuts in any graph G must be at least $\frac{OPT_G}{2}$. Due to **the probabilistic method**, in particular **the averaging principle**, there must exists a bipartition of V into S and T whose cut $E(S,T)$ is of size at least $\frac{OPT_G}{2}$. Then next question is how to find such a bipartition $\{S,T\}$ *algorithmically*.

We still fix an arbitrary order of all vertices as $V = \{v_1, v_2, \ldots, v_n\}$. Recall that each vertex v_i is associated with a uniform and independent random bit X_{v_i} to indicate whether v_i joins S or T. We want to fix the value of X_{v_i} one after another to construct a bipartition $\{\hat{S}, \hat{T}\}$ of V such that

$$|E(\hat{S},\hat{T})| \geq \mathbb{E}[|E(S,T)|] \geq \frac{OPT_G}{2}.$$

We start with the first vertex v_i and its random variable X_{v_1}. By the **law of total expectation**,

$$\mathbb{E}[E(S,T)] = \tfrac{1}{2}\mathbb{E}[E(S,T) \mid X_{v_1} = 0] + \tfrac{1}{2}\mathbb{E}[E(S,T) \mid X_{v_1} = 1].$$

There must exist an assignment $x_1 \in \{0,1\}$ of X_{v_1} such that

$$\mathbb{E}[E(S,T) \mid X_{v_1} = x_1] \geq \mathbb{E}[E(S,T)].$$

We can continuously applying this argument. In general, for any $i \leq n$ and any particular partial assignment $x_1, x_2, \ldots, x_{i-1} \in \{0,1\}$ of $X_{v_1}, X_{v_2}, \ldots, X_{v_{i-1}}$, by the law of total expectation

$$\begin{aligned}\mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_{i-1}} = x_{i-1}] = {} & \frac{1}{2}\mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_{i-1}} = x_{i-1}, X_{v_i} = 0] \\ & + \frac{1}{2}\mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_{i-1}} = x_{i-1}, X_{v_i} = 1].\end{aligned}$$

There must exist an assignment $x_i \in \{0,1\}$ of X_{v_i} such that

$$\mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_i} = x_i] \geq \mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_{i-1}} = x_{i-1}].$$

By this argument, we can find a sequence $x_1, x_2, \ldots, x_n \in \{0,1\}$ of bits which forms a *monotone path*:

$$\mathbb{E}[E(S,T)] \leq \cdots \leq \mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_{i-1}} = x_{i-1}] \leq \mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_i} = x_i] \leq \cdots \leq \mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_n} = x_n].$$

We already know the first step of this monotone path $\mathbb{E}[E(S,T)] \geq \frac{OPT_G}{2}$. And for the last step of the monotone path $\mathbb{E}[E(S,T) \mid X_{v_1} = x_1, \ldots, X_{v_n} = x_n]$ since all random bits have been fixed, a bipartition $(\hat{S}, \hat{T})$ is determined by the assignment $x_1, \ldots, x_n$, so the expectation has no effect except just retuning the size of that cut $|E(\hat{S}, \hat{T})|$. We found the cut $E(\hat{S}, \hat{T})$ such that $|E(\hat{S}, \hat{T})| \geq \frac{OPT_G}{2}$.

We translate the procedure of constructing this monotone path of conditional expectation to the following algorithm.

MonotonePath

Input: undirected graph $G(V, E)$,

with an arbitrary order of vertices $V = \{v_1, v_2, \ldots, v_n\}$;

---

initially $S = T = \emptyset$;
for $i = 1, 2, \ldots, n$

v_i joins one of S, T to maximize the average size of cut conditioning on the choices made so far by the vertices $v_1, v_2, \ldots, v_i$;

We leave as an exercise to verify that the choice of each v_i (to join which one of S, T) in the *MonotonePath* algorithm (which maximizes the average size of cut conditioning on the choices made so far by the vertices $v_1, v_2, \ldots, v_i$) must be the same choice made by v_i in the *GreedyMaxCut* algorithm (which maximizes the current $|E(S,T)|$).

Therefore, the greedy algorithm for max-cut is actually due to a derandomization of average-case.

Derandomization by pairwise independence

We still construct a random bipartition of V into S and T. But this time the random choices have **bounded independence**.

For each vertex $v \in V$, we use a Boolean random variable $Y_v \in \{0,1\}$ to indicate whether v joins S and T. The dependencies between Y_v's are to be specified later.

By linearity of expectation, regardless of the dependencies between Y_v's, it holds that:

$$\mathbb{E}[|E(S,T)|] = \sum_{uv \in E} \Pr[Y_u \neq Y_v].$$

In order to have the average cut $\mathbb{E}[|E(S,T)|] = \frac{|E|}{2}$ as the fully random case, we need $\Pr[Y_u \neq Y_v] = \frac{1}{2}$. This only requires that the Boolean random variables Y_v's are uniform and **pairwise independent** instead of being **mutually independent**.

The n pairwise independent random bits $\{Y_v\}_{v \in V}$ can be constructed by at most $k = \lceil \log(n+1) \rceil$ mutually independent random bits $X_1, X_2, \ldots, X_k \in \{0,1\}$ by the following standard routine.

Theorem

Let $X_1, X_2, \ldots, X_k \in \{0,1\}$ be mutually independent uniform random bits.
Let $S_1, S_2, \ldots, S_{2^k-1} \subseteq \{1,2,\ldots,k\}$ enumerate the $2^k - 1$ nonempty subsets of $\{1,2,\ldots,k\}$.
For each $i \leq i \leq 2^k - 1$, let

$$Y_i = \bigoplus_{j \in S_i} X_j = \left(\sum_{j \in S_i} X_j\right) \bmod 2.$$

Then $Y_1, Y_2, \ldots, Y_{2^k-1}$ are pairwise independent uniform random bits.

If Y_v for each vertex $v \in V$ is constructed in this way by at most $k = \lceil \log(n+1) \rceil$ mutually independent random bits $X_1, X_2, \ldots, X_k \in \{0,1\}$, then they are uniform and pairwise independent, which by the above calculation, it holds for the corresponding bipartition $\{S,T\}$ of V that

$$\mathbb{E}[|E(S,T)|] = \sum_{uv \in E} \Pr[Y_u \neq Y_v] = \frac{|E|}{2}.$$

Note that the average is taken over the random choices of $X_1, X_2, \ldots, X_k \in \{0,1\}$ (because they are the only random choices used to construct the bipartition $\{S,T\}$). By the probabilistic method, there must exist an assignment of $X_1, X_2, \ldots, X_k \in \{0,1\}$ such that the corresponding Y_v's and the bipartition $\{S,T\}$ of V indicated by the Y_v's have that

$$|E(S,T)| \geq \frac{|E|}{2} \geq \frac{OPT}{2}.$$

This gives us the following algorithm for exhaustive search in a smaller solution space of size $2^k - 1 = O(n^2)$.

Algorithm

Enumerate vertices as $V = \{v_1, v_2, \ldots, v_n\}$;
let $k = \lceil \log(n+1) \rceil$;
for all $\vec{x} \in \{0,1\}^k$

initialize $S_{\vec{x}} = T_{\vec{x}} = \emptyset$;
for $i = 1, 2, \ldots, n$

if $\bigoplus_{j:\lfloor i/2^j \rfloor \bmod 2=1} x_j = 1$ then v_i joins $S_{\vec{x}}$;
else v_i joins $T_{\vec{x}}$;

return the $\{S_{\vec{x}}, T_{\vec{x}}\}$ with the largest $|E(S_{\vec{x}}, T_{\vec{x}})|$;

The algorithm has approximation ratio 1/2 and runs in polynomial time.
