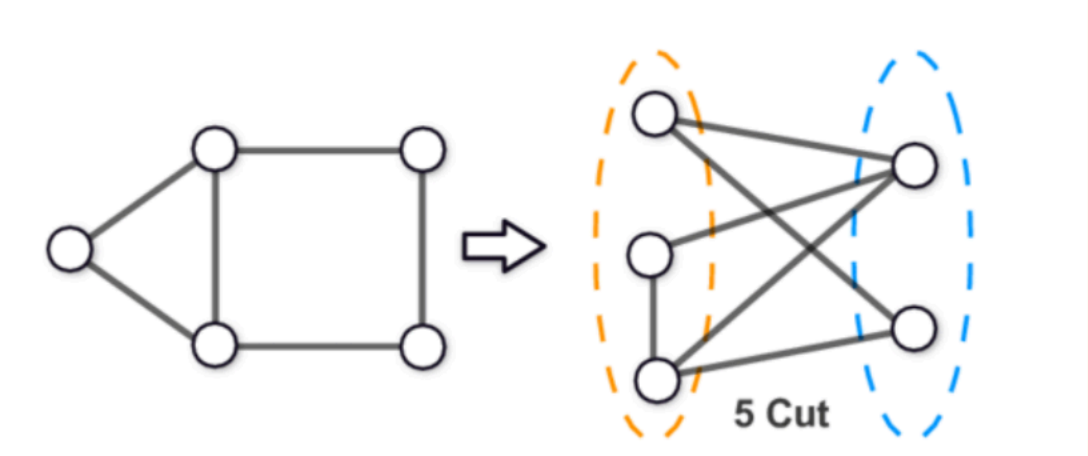
An Experimental Analysis of Max-Cut Algorithms

Andrew Morato

The Max-Cut Problem

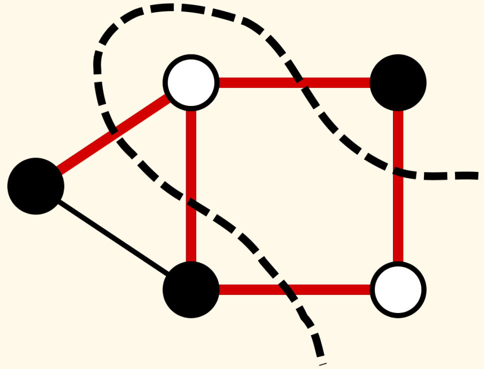
- Similar to the min-cut problem that is solvable by the Ford-Fulkerson Algorithm
- Given an undirected graph G = (V, E) with a positive integer weight on each edge, find a partition [A, B] (or coloring) of the vertex set such that the weight of all edges with one end in A and the other in B is maximized
- Alternatively, we want to find a bipartite subgraph of G where the edges' combined weight is maximized



Example of a partitioned vertex set with a cut of 5

Hardness and Challenges of the Max-Cut Problem

- Unlike the min-cut problem that is solvable in polynomial time, there are no known polynomial time algorithms for the max-cut problem
- In some graphs with certain attributes, the max-cut problem can be solved in polynomial time
  - For example, in planar graphs, graphs where edges intersect only at the vertices of the graph, a cut with maximum weight can be found in polynomial time
- The max-cut problem is NP-hard



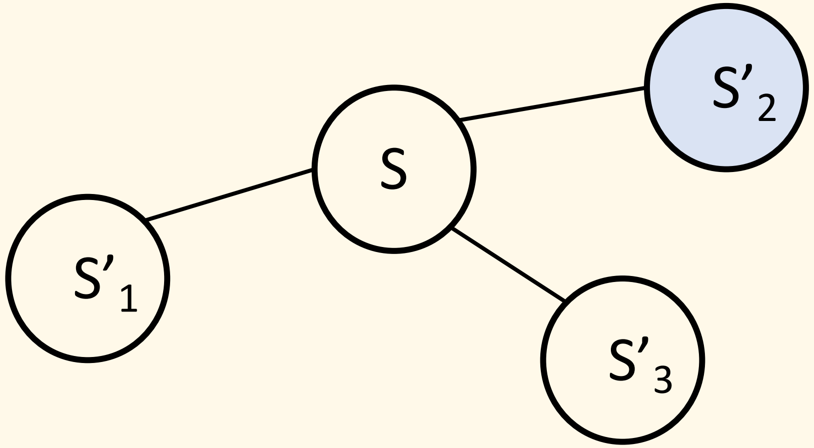
Approximation Algorithms to Find the Max-Cut

- Since there are no known polynomial time solutions to max-cut, in this experimental analysis, we consider two approaches to approximate the maximum cut in a graph,
- Approximation of the max-cut via a **local search** method
  - Proposed in Chapter 12 of *Algorithm Design* by Kleinberg and Tardos
  - The vertex set is divided into two partitions and we progressively flip a vertex from one partition to the other if it increases the cost of the cut
- Approximation of the max-cut via a **greedy approach**
  - Proposed in *Yet Another Algorithm for Dense Max Cut: Go Greedy* by Balcan, Blum, and Vempala
  - A greedy approach where we consider the vertices in random order and assign them to a partition based on which partition increases the cost of the cut

Brief Overview of Local Search

- Local search is a technique that describes algorithms that explore the space of possible solutions to a problem sequentially, moving from one solution to a "nearby" one. The idea is to move to better and better solutions to eventually find an optimal one. There are two main components of a local search solution,
  - **Neighbor Relation** – Other neighboring solutions S' would be considered "nearby" if those solutions fall within this neighbor relation with the current solution S
  - **Choice of Neighboring Solution** – Once the neighbors are defined, the algorithm chooses a "better" neighbor S' of S (within the neighborhood of S as defined by its neighbor relation) and continues to the next iteration

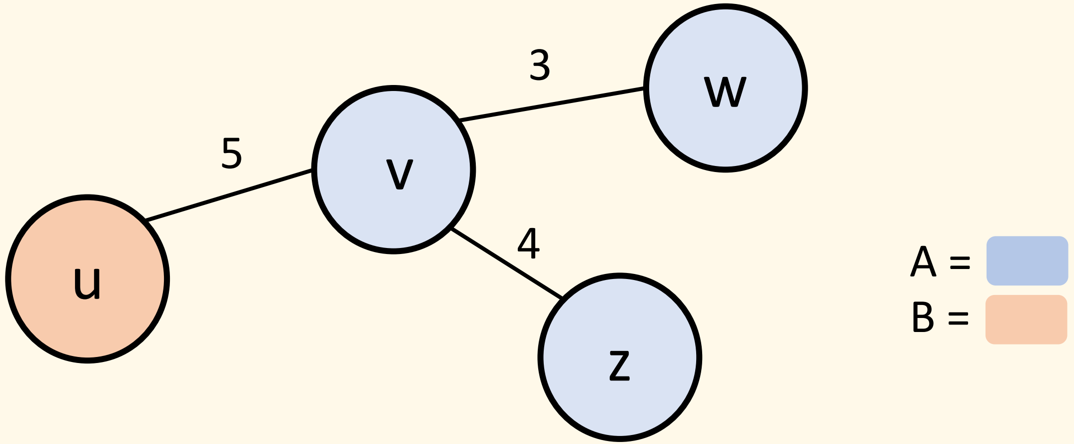
In the example to the right, S would be the current solutions and S'_1 S'_2 and S'_3 are the neighboring solutions, where S'_2 is the chosen neighbor



Max-Cut via Local Search

- The goal is to find a partition [A, B] of the vertex set such that the weight of all edges with one end in A and the other in B is maximized
- In such a partition, if there is a vertex v such that the total weight of the edges from v to vertices on its own side of the partition exceeds the total weight of the edges from v to vertices on the other side of the partition, then v should be flipped to the other side of the partition
- This is known as a single-flip. Therefore, in this algorithm, the neighborhood of a solution S would be solutions that differ from S by just a single-flip of any one vertex
- At each iteration, we would choose the neighboring solution with the best flip

In this example, the weight of edges from v to vertices in A (z & w) exceeds the weight from v to vertices in B (u), so v should be flipped from A to B



Max-Cut via Greedy Approach

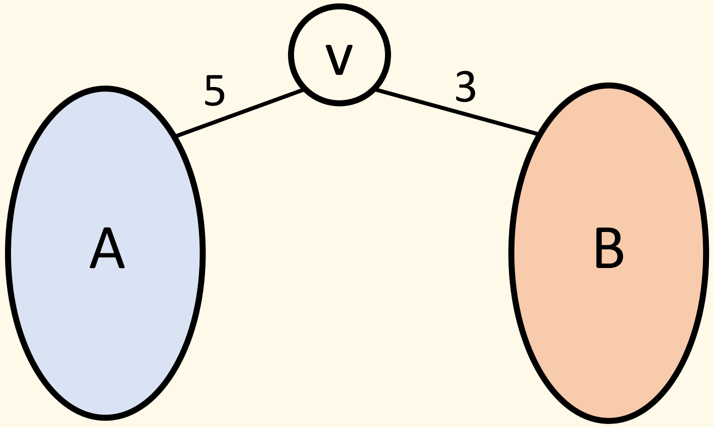
- The naïve greedy approach picks the best choice now
- This greedy approach considers vertices one by one in random order and places them in partition A or partition B depending on the edge weights to neighbors that are already in one of the two partitions
- A proposed variation on this algorithm repeats this process several times and chooses the best cut found
  - If repeated enough times $2^{2^{O(poly(n))}}$, we could guarantee near-optimality
  - The variation is a simple algorithm, albeit quite slow

ALGORITHM

Repeat $2^{2^{O(poly(n))}}$ times

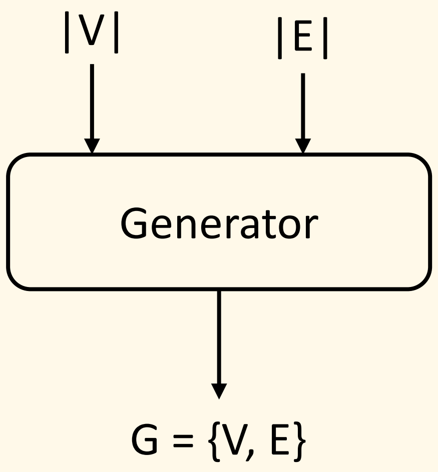
For each vertex v in V in random order,

Place v on the side that maximizes the number of resulting crossing edges



Experimental Setting and Data Source

- In order to use graphs of arbitrary size and features for the experimentation and testing of both algorithms, I implemented a graph generator
- The generator takes the number of vertices and edges creates a random graph G = {V, E} with those specifications
- Scripts produce four datasets (containing ~1000 graphs each) that have the following characteristics,
    - Dataset 1: Vertices have high degree
    - Dataset 2: Vertices have low degree
    - Dataset 3: High count of vertices (> 1000)
    - Dataset 4: Low count of vertices (< 1000)



References

Here are the references that I consulted in researching for this project,

- Chapter 12 in *Algorithm Design* by Jon Kleinberg and Eva Tardos
- *Yet Another Algorithm for Dense Max Cut: Go Greedy* by Maria-Florina Balcan, Avrim Blum, and Santosh Vempala
- Additional lecture on maximum cuts: https://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec02.pdf
- On planar graphs: http://discrete.openmathbooks.org/dmoi3/sec_planar.html