Wave Function Collapse Partitioning: A New Approach to Maximum Cut

Artem Gryniv
Saint Viator High School
Arlington Heights, IL
artem.gryniv@icloud.com

David Perkins
Department of Computer Science
Colgate University, Hamilton, NY
dperkins@colgate.edu

Abstract

We propose a competitive metaheuristic approach to the Maximum Cut problem based on the Wave Function Collapse algorithm, called Wave Function Collapse Partitioning (**WFC-P**). Maximum Cut is an NP-Complete combinatorial optimization problem rooted in graph theory with applications in efficient network design and machine learning. Due to its inherent computational complexity, exact algorithms are impractical, while common heuristics are constrained by local optima. **WFC-P** overcomes these limitations with each iteration of the algorithm consisting of three phases: vertex selection by entropy, subset assignment through Simulated Annealing, and forced partitioning of vertices through propagation. Extensive testing on various libraries and graph structures indicates that **WFC-P** consistently outperforms local search and greedy heuristics and produces competitive results with the best-known approximation algorithm. Similar adaptations of Wave Function Collapse and metaheuristics can be applied to other NP-Hard and NP-Complete problems, potentially leading to transformative solutions in numerous real-world scenarios.

1 Introduction

The Maximum Cut problem (Max Cut) is a combinatorial optimization problem in graph theory involving the partitioning of a graph into two disjoint subsets of vertices to maximize the number of shared edges, commonly referred to as the *cuts* in the graph. Often, the edges are assigned a weight, modifying the goal of the problem to be maximizing the total weight of the cut edges. The total weight of the cut is referred to as the *cut value*. The weighted Max Cut problem will be the study of the rest of this paper.

Unlike its counterpart Minimum Cut, which can be solved in polynomial time using the Edmonds-Karp algorithm [2], the optimization version of Max Cut was one of the first problems proven to be NP-Complete [4]. Max Cut has applications in efficient electrical circuit and communication network design [7]. Also, because the problem involves partitioning a graph into two well-separated subsets, it naturally applies to binary classification in machine learning.

While exact approaches to Max Cut exist, they are computationally expensive and only feasible for small and moderately sized graphs. For example, a brute force algorithm involves evaluating all possible partitions of a graph's vertices into two subsets, resulting in a time complexity of $O(2^n)$. Due to the exponential time complexity of exact algorithms, researchers have turned to approximation and heuristic algorithms for Max Cut. While these algorithms do not guarantee the best answer, they are fast and can partition larger graphs in a feasible amount of time.

Two popular heuristics used for many other NP-Hard and NP-Complete combinatorial problems are *Greedy* and *Local Search*, both with an approximation ratio of $1/2$. The implementation of each of these heuristics is outlined in [1] for Max Cut. The best known approach for Max Cut (the Goemans-Williamson algorithm proposed in [3]) is a global approximation algorithm with a 0.87856 approximation ratio. We will compare our new partitioning heuristic to these three established algorithms.

Our new heuristic for Max Cut, named Wave Function Collapse Partitioning (**WFC-P**), is inspired by the wave function collapse procedure in quantum mechanics. We apply Simulated Annealing to guide **WFC-P** as it partitions the graph.

The structure of the paper is as follows: in Section 2, we discuss the formal definition of Max Cut; in Section 3, we examine established Max Cut algorithms; in Section 4, we formally present **WFC-P**; in Section 5, we compare the performance of **WFC-P** with the established algorithms; in Section 6, we formally con-

clude our paper with a discussion of further potential research.

2 Max Cut Problem Description

Given an undirected graph $G = (V, E)$ with a vertex set $V = \{1, 2, \ldots, n\}$ and an edge set $E \subseteq V \times V$, where each edge $(i, j) \in E$ has a non-negative weight $w_{ij} = w_{ji}$, we *cut* the vertex set V into two disjoint subsets S and T such that the total weight of the edges with one endpoint in S and the other in T is maximized. The *cut value* is the sum of the weights of the edges in the cut, or

$$w(S, T) = \sum_{i \in S, j \in T} w_{ij}.$$

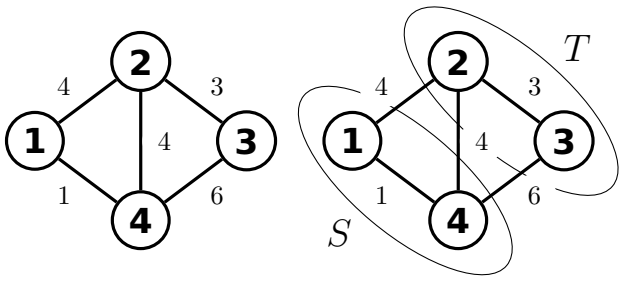

Figure 1: A simple weighted graph (left) and a partition into sets S and T for Max Cut (right).

Figure 1 shows a simple weighted graph on the left and a cut on the right. A *cut edge* is defined as an edge that connects a vertex from S to a vertex in T, so the cut edges are $\{(1, 2), (4, 2), (4, 3)\}$. The cut value of this (maximum) cut is 14.

3 Famous MC algorithms

In this section, we discuss the three algorithms against which we will test **WFC-P**: *Greedy*, *Local Search*, and *Goemans-Williamson*.

3.1 Greedy

The *Greedy* approach, outlined in *Algorithm 1*, sorts the vertices by degree (from highest to lowest) and iteratively assigns each vertex to the subset S or T that maximizes $w(S, T)$. It can partition a graph in linear time with a time complexity of $O(V \cdot E)$.

Algorithm 1: *Greedy*

Input : A simple connected graph $G = (V, E)$ and nonnegative edge weights $w_{ij} = w_{ji}$

Output: Two subsets S and T of V that attempt to maximize $w(S, T)$

```
w ← highest degree first ordering of V
for vertex v in w do
    place v in the subset that maximizes w(S, T)
end
```

3.2 Local Search

The *Local Search* approach, outlined in *Algorithm 2*, involves starting with a randomly partitioned graph. It then checks if moving a vertex from one subset to the other increases the cut value. If so, it will perform the move. *Local Search* will run until no moves can be made to improve the cut value, resulting in an algorithm that will run in linear time in the best case and quadratic time in the worst case.

Algorithm 2: *Local Search*

Input : A simple connected graph $G = (V, E)$ and nonnegative edge weights $w_{ij} = w_{ji}$

Output: Two subsets S snd T of V that attempt to maximize $w(S, T)$

```
randomly partition V into subsets S and T
repeat until no improvement can be made
    for vertex v in V do
        S ← the subset containing v
        T ← the other subset
        if moving v to T increases w(S, T) then
            move v to T
        end
    end
end
```

3.3 Goemans-Williamson

The Goemans-Williamson algorithm [3] uses semidefinite programming (SDP) followed by a randomized rounding procedure to guarantee a solution that is at least 87.856% of the optimal solution. It begins by transforming the Max Cut problem into a quadratic optimization problem, which can be represented by the maximization of the following expression:

$$\frac{1}{2} \sum_{i<j} w_{ij}(1 - \mathbf{v}_i \cdot \mathbf{v}_j)$$

where $w_{ij} = w_{ji}$ are non-negative weights for each pair of vertices i and j, and $\mathbf{v}_i \cdot \mathbf{v}_j$ represents the dot product between $\mathbf{v}_i$ and $\mathbf{v}_j$. The algorithm then relaxes the

original problem by allowing the vectors $\mathbf{v}_i$ to be arbitrary unit vectors in $\mathbb{R}^n$, where $\mathbb{R}^n$ represents the n-dimensional Euclidean space of real numbers. This relaxation is then solved in polynomial time using SDP to obtain an optimal set of vectors $\mathbf{v}_i$. These vectors represent the optimal solution for the relaxed problem but are not necessarily a valid solution to the original Max Cut problem. To obtain a valid cut, a hyperplane through the origin in $\mathbb{R}^n$ is chosen, and each vector is mapped to one of the two sides of the hyperplane. For any given set of vectors $\mathbf{v}_i \in S_n$, the expected weight of the cut defined by the random hyperplane is

$$\mathbb{E}[W] = \sum_{i<j} w_{ij} \left(\frac{\arccos(\mathbf{v}_i \cdot \mathbf{v}_j)}{\pi} \right)$$

This expected value is guaranteed to be at least 87.856% of the maximum possible cut value. It is conjectured among researchers that this value may be the best possible approximation, making the Goemans-Williamson algorithm a powerful approach to Max Cut.

4 Wave Function Collapse Partitioning

Our new Max Cut heuristic, Wave Function Collapse Partitioning (**WFC-P**), follows the structure of the procedural generating algorithm Wave Function Collapse (WFC). Given an initial series of constraints, WFC generates procedural patterns from a sample image [8]. The algorithm iterates through three phases: observe, collapse, and propagate. WFC begins by observing a grid and locating the most restricted tile. It then collapses the tile by assigning it a pattern from the sample image and propagates the effect of this assignment by restricting the possible patterns available to the remaining affected tiles.

Our approach to Max Cut also involves the implementation of Simulated Annealing, a metaheuristic analogous to the way heated metals cool and anneal in thermodynamics [9]. First proposed by Scott Kirkpatrick [5], Simulated Annealing allows heuristic algorithms to make suboptimal decisions that deviate from their natural inclination. Since heuristics are deterministic and often get stuck at local optima, Simulated Annealing is used to promote exploration of a problem by accepting "bad" moves. Making these suboptimal decisions may allow the heuristic to break out of local optima and continue toward the global optimum. The probability of a heuristic making this suboptimal choice is called the acceptance probability and is a function of the temperature, which decreases by some factor after each iteration of the algorithm. Therefore, the acceptance probability decreases as the heuristic progresses, resulting in an algorithm that first explores the problem before committing to a solution path. Pseudocode for **WFC-P** begins in *Algorithm 3*.

Algorithm 3: WFC-P: Wave Function Collapse Partitioning

Input : A simple connected graph $G = (V, E)$, nonnegative edge weights $w_{ij} = w_{ji}$, an initial temperature t, a percentage P, and a constant $a > 0$

Output: Two subsets of V that maximize total weight of cut edges

```
v ← the vertex with highest degree
place v in a subset
propagate(v)
while there are unpartitioned vertices do
    observe(G)
    collapse(v, t, a)
    propagate(v, G)
    t = t · P
end
output the proper partition
```

WFC-P involves three functions ($observe()$, $collapse()$, and $propagate()$), representing the three phases of WFC. The simple, undirected input graph we run **WFC-P** on can be seen as the grid WFC works on, with vertices representing the tiles. A vertex is considered partitioned once it has been moved to a subset. We define the *entropy* of a vertex to be the sum of its weighted edges that connect to its partitioned neighbors. Unlike how WFC selects the vertex with lowest entropy, our heuristic will select vertices with highest entropy. Since our goal is to maximize the weight of the cut, we must first select vertices with stronger connections to each subset in order to allow our propagate function to increase the entropy of the remaining vertices with lower entropy. This way, we are holding off on collapsing vertices with weak connections and allowing them to grow in strength.

Let $S = \{\}$ and $T = \{\}$ be the two subsets of vertices. We make S and T global variables. We use $t = 200$, $P = 0.95$, and $a = 200$ for the Simulated Annealing process. We begin our algorithm by placing the vertex with the highest degree into a subset, breaking ties at random. We then propagate this effect by calling $propagate()$ (see *Algorithm 6)*. Finally, we iterate through our three functions until all vertices have been partitioned. We decrease t after each iteration of our algorithm by multiplying it by P.

The function $observe()$ (see *Algorithm 4)* returns the vertex with highest entropy. If there are any isolated vertices, then $observe()$ will move them to either set, as they will not affect the cut value.

Algorithm 4: observe(G)

Input : A graph $G = (V, E)$ and nonnegative edge weights $w_{ij} = w_{ji}$
Output: The vertex v with highest entropy

$v \leftarrow$ the vertex with highest entropy
if *entropy*$(v) = 0$ **then**
  place v and the remaining unpartitioned vertices in either set
  terminate the algorithm
end
return v

The function $collapse()$ (see *Algorithm 5*) assigns the vertex selected by $observe()$ to a subset. It begins by calculating cut value when the vertex is placed in each subset and finds the subset that resulted in a greater value. We apply simulated annealing in this function to select which subset to place the vertex in. The inequality $r < exp(-a/t)$ represents the probability that our algorithm places the vertex in the subset that resulted in a smaller cut value—a sub-optimal decision.

Algorithm 5: collapse(v, t, a)

Input : The vertex v selected by $observe()$, a temperature t, and a constant $a > 0$

calculate the cut value when v is placed in each subset
$s \leftarrow$ the subset that resulted in a larger cut value
$m \leftarrow$ the other subset
$r \leftarrow$ a random float between 0 and 1
if $r < exp(-a/t)$ **then**
  place v in m
end
else
  place v in s
end

The function $propagate()$ determines if the neighbors of the collapsed vertex can be forced into a subset. For each neighbor, the algorithm checks whether all of its neighbors have been assigned to the same subset. If so, the collapsed vertex's neighbor is assigned to the subset opposite from the subset its neighbors are in. The $propagate()$ function then updates the entropy of each of the collapsed vertex's neighbors.

Algorithm 6: propagate(v, G)

Input : A graph $G = (V, E)$, nonnegative edge weights $w_{ij} = w_{ji}$, and a partitioned vertex v

for each unpartitioned neighbor n of v **do**
  if all of n's neighbors are partitioned and in the same set **then**
    move n to the other set
  end
end
update the entropy of v's unpartitioned neighbors

5 Experimental Results

In this section, we compare **WFC-P** to the algorithms described in Section 3: *Greedy*, *Local Search*, and Goemans-Williamson (GW). We implemented **WFC-P**, *Greedy*, and *Local Search* in Julia 1.10.3 and ran tests on an Apple M3 Max processor. To test the performance of **WFC-P**, we used instances from a Max Cut and BQP instance library [6], from the symmetric traveling salesman problem (TSP) input graphs from TSPLIB [10], and from our own randomly generated unweighted graphs with various properties. We ran *Greedy*, *Local Search*, and **WFC-P** 100,000 times on each graph and reported the best cut value achieved.

5.1 On BQP instances

Table 1 displays the comparative results between **WFC-P**, *Local Search*, and *Greedy* on twelve weighted graphs, which can be found the instance library [6]. $|V|$ and $|E|$ represent the number of vertices and edges in each graph respectively. C^* is the optimal cut value reported by the instance library. The algorithms and their adjacent time columns indicate the best cut value reported by each algorithm and the median time (in milliseconds) they took to partition the graph.

WFC-P found the optimal cut value on the instance `mannino_k48` and a better cut value than both *Greedy* and *Local Search* on the other eleven instances. On average, **WFC-P** ran 3.25 times slower than *Local Search* and 4.44 times slower than *Greedy*. However, since the run time of each algorithm scales linearly, **WFC-P** will continue to be roughly 3.25 and 4.44 times slower as the size of the graph increases.

Table 1: On BPQ graphs: C^* is the optimal cut value reported by the library, and all times are in milliseconds

Name	$\|V\|$	$\|E\|$	C^*	LS	Time	Greedy	Time	WFC-P	Time
mannino_k48	48	1,128	252,518,838	252,518,838	0.143	249,046,909	0.116	252,518,838	0.513
mannino_k487b	48	5,391	3,655,475	3,550,070	0.970	3,550,295	0.710	3,643,170	3.662
mannino_k487c	48	8,511	8,640,860	8,570,460	1.700	8,279,675	1.253	8,573,295	5.724
pw01_100.0	100	495	2,019	1,914	0.082	1,892	0.067	1,986	0.252
pw01_100.4	100	495	2,039	1,947	0.088	1,939	0.065	1,995	0.256
pw01_100.8	100	495	2,022	1,886	0.081	1,913	0.067	1,970	0.251
pw05_100.1	100	2,475	8,045	7,889	0.387	7,716	0.267	7,996	1.146
pw05_100.5	100	2,475	8,169	7,910	0.358	7,934	0.273	8,028	1.161
pw05_100.9	100	2,475	8,099	7,966	0.422	7,930	0.267	7,993	1.200
pw09_100.2	100	4,455	13,461	13,315	0.574	13,038	0.434	13,394	2.087
pw09_100.3	100	4,455	13,656	13,419	0.642	13,384	0.447	13,515	2.094
pw09_100.6	100	4,455	13,640	13,498	0.700	13,371	0.443	13,508	2.286

5.2 On unweighted graphs

In Table 2, we compare **WFC-P**, *Greedy*, and *Local Search* on unweighted graphs with various properties. All of edges in the graphs have a weight of 1, so the goal of each algorithm is to partition the vertices to maximize the number of cut edges. The graphs were randomly generated based on property inputs. Run times are not compared here as they follow the same proportions in Table 1. $|V|$ represents the number of vertices in the graph. The largest number of cut edges produced by each algorithm is reported in their respective columns.

Table 2: **WFC-P**, LS, and *Greedy* on Unweighted Graphs

Graph	$\|V\|$	LS	Greedy	**WFC-P**
Cluster (3)	20	57	57	57
Cluster (5)	100	723	725	737
Cluster (10)	1,000	47,592	47,522	47,611
Cycle	1,000	1,000	1,000	1,000
Cycle	1,001	1,000	1,000	1,000
Regular (3)	20	27	27	27
Regular (10)	2,000	6,696	6,572	6,860
Dense	10	23	23	24
Dense	100	2,330	2,332	2,338
Dense	500	57,146	57,149	57,204
Complete	100	2,500	2,500	2,500
Complete	1,001	250,500	250,500	250,500

In the graph column of the table, we present five types of graphs: cluster, cycle, regular, dense, and complete. Cluster graphs are graphs in which groups of vertices (clusters) are formed with minimal shared edges between each. The number in parentheses following the graph type indicates the number of clusters. In cycle graphs, every vertex has two neighbors, forming one large circle of vertices. Regular graphs consist of all vertices having the same degree. The number in parentheses following the graph type indicates the degree of each vertex. In dense graphs, nearly every vertex is connected to every other. Similar to dense graphs, a complete graph is one in which every vertex is connected to every other. While these two graphs have similar definitions, the results produced by the various algorithms contrast significantly.

On both cluster and regular graphs, **WFC-P** finds the same cut value as LS and *Greedy* on smaller instances. However, on larger graphs, **WFC-P** outperforms both algorithms. For all cycle graphs, **WFC-P**, LS, and *Greedy* always found the optimal cut, which is given by $|V|$ if $|V|$ is even and $|V|$ - 1 if $|V|$ is odd. In dense graphs, **WFC-P** consistently outperformed LS and *Greedy* on smaller and larger graphs. All three algorithms were able to find the optimal cut value for complete graphs, which is given by $\left\lfloor \frac{n^2}{4} \right\rfloor$.

5.3 Comparing WFC-P against GW

Table 3 displays the results of **WFC-P** and GW on TSP graphs. $|V|$ displays the number of cities (vertices) in the graph, and the GW and **WFC-P** columns display the cut value produced by each respective algorithm. The selected graphs and GW results were obtained from the results table presented by Michel X. Goemans and David P. Williamson in [3]. It is important to note that the run times of each algorithm were computed on different processors, so the true differences may vary. However, it is evident that **WFC-P** partitions the graphs much more quickly than GW.

In six out of nine test graphs, **WFC-P** output the same cut value as GW. In the remaining three graphs (hk48, kroD100, and kroE100) **WFC-P** produced a cut value that was 99.66%, 99.91%, and 99.98% of the value produced by GW respectively. In all nine test graphs, GW produced the optimal solution, meaning our algorithm produced the optimal solution for six graphs and was within 99.5% of the optimal solution for the remaining

Table 3: Comparing **WFC-P** against GW

Name	$\|V\|$	GW	Time(s)	**WFC-P**	Time(s)
dantzig42	42	42,638	43.35	42,638	0.000357
gr120	120	2,156,667	754.87	2,156,667	0.003225
gr48	48	320,277	26.17	320,277	0.000452
hk48	48	771,712	66.52	769,099	0.000469
kroA100	100	5,897,392	420.83	5,897,392	0.002210
kroB100	100	5,763,047	917.47	5,763,047	0.002246
kroC100	100	5,890,760	398.78	5,890,760	0.002250
kroD100	100	5,463,250	469.48	5,458,390	0.002252
kroE100	100	5,986,591	375.68	5,985,610	0.002229

three. We consider these to be successful results as it is conjectured that the approximation ratio of Goemans-Williamson algorithm is likely unbeatable.

6 Conclusion and Further Research

In this paper, we presented a Wave Function Collapse inspired Max Cut heuristic guided by Simulated Annealing. The implementation of our metaheuristic allows **WFC-P** to escape the local optima where many of the current established heuristics get stuck in. Through extensive testing on various instance graphs, we demonstrated that **WFC-P** consistently outperforms the *Local Search* and *Greedy* algorithms in finding the Max Cut. We also demonstrated that **WFC-P** yeilds results competitive to those of the best-known Max Cut algorithm, Goemans-Williamson, achieving a cut value of at least 99.5% of optimal solution.

We believe our approach holds significant potential for future research as it can be extended to other NP-Hard and NP-Complete combinatorial optimization problems. The original Wave Function Collapse algorithm, designed for generating procedural images from a sample image given a series of constraints, has been adapted and enhanced with Simulated Annealing to create a competitive algorithm for the extensively researched Maximum Cut problem. We encourage the exploration of other metaheuristics combined with a variation of WFC, as they could potentially lead to innovative solutions for a wide range of optimization problems.

References

[1] Haowen Chan. 15-854: Approximation algorithms: Max-cut, hardness of approximations. Lecture notes. Instructor: Anupam Gupta, September 14.

[2] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computing Machinery*, 19(2), April 1972. [Accessed: August 16, 2024].

[3] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Complexity of Computer Computations*, 42(6):1115–1145, November 1995. [Accessed: August 24, 2024].

[4] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 1972. [Accessed: August 18, 2024].

[5] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimizations by simulated annealing. *Science*, 220(4598), May 1983. [Accessed: August 22, 2024].

[6] Sven Mallach. Instance library for the maximum cut and the unconstrained binary quadratic optimization problem, 2021. Accessed: September 8, 2024.

[7] Sum of Squares. Maximum cut and related problems. `https://www.sumofsquares.org/public/lec02-1_maxcut.pdf`. [Accessed: August 12, 2024].

[8] Joseph Parker. Generating worlds with wave function collapse. *PROCJAM Tutorials*, 15(3):23–30, 2017. [Accessed: August 15, 2024].

[9] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C, Second Edition*. Cambridge University Press, 1992. See Chapter 10, pp. 444-445.

[10] Gerhard Reinelt. TSPLIB – A Traveling Salesman Problem Library. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB`, 1991. Accessed: September 8, 2024.