

Hadlock's Algorithm for Planar Max-Cut

From Theory to Python to C++

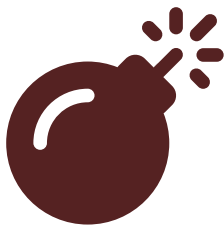
30-min Talk · Sisyphus · 2026-05-11

1. Problem: Maximum Cut



Given: Graph $G = (V, E)$ with edge weights $w : E \rightarrow \mathbb{R}^+$

Find: Partition $V = S \cup (V \setminus S)$ maximizing weight of crossing edges

$$\text{Max-Cut}(G) = \max_{S \subseteq V} \sum_{\substack{(u,v) \in E \\ u \in S, v \notin S}} w(u,v)$$



Syntax error in text
mermaid version 11.14.0

 Crossing edges = **cut weight** (green),  Internal edges = **not cut**

2. Why Is This Hard?

- **NP-complete** for general graphs (Karp's 21 problems, 1972)
- No poly-time algorithm unless $P = NP$
- Best general approximation: **0.878-approximation** (Goemans–Williamson, SDP)

But... Planar Graphs are Special 📖

Property	General Graph	Planar Graph
Max-Cut complexity	NP-complete	Polynomial 🎉
Edge count	$O(V^2)$	$O(V)$
Dual graph	X	Exists

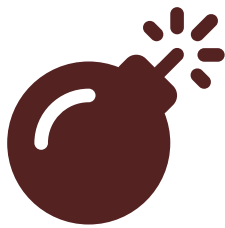
Hadlock (1975) discovered: planarity makes Max-Cut tractable.

3. Key Insight 💡

Odd cycles are the enemy of bipartiteness.

A cut is bipartite \Leftrightarrow it contains no odd cycles.

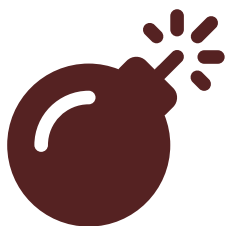
The Duality Trick



Syntax error in text
mermaid version 11.14.0

Each face in $G \rightarrow$ vertex in G^* . Edge weight = shared primal edge weight.

4. Algorithm Flow ↻



Syntax error in text
mermaid version 11.14.0

6 steps. $O(n^3)$. Exact for planar graphs.

5. Step 1: Dual Graph Construction

```
def build_dual(faces, weight):
    # Edge → incident face indices
    edge_face_map = {}
    for fi, face in enumerate(faces):
        for (u, v) in face_edges(face):
            edge_face_map[ordered(u, v)].append(fi)

    # Adjacent face pairs → min weight edge
    for primal, face_ids in edge_face_map.items():
        for fi, fj in pairs(face_ids):
            best[(fi, fj)] = min(best.get(...), weight(primal))
```

- Each face becomes a dual vertex
 - Adjacent faces connected by weighted edge (min weight for parallel edges)
 - Result: sparse graph with $O(F)$ edges ($F = \#$ faces)
-

6. Step 2: Odd Faces

```
odd_faces = [i for i, face in enumerate(faces)
              if len(face) % 2 == 1]
```

- **Odd face** = face with odd number of boundary edges
- In triangulated planar graphs: every face is a triangle → all faces odd
- The number of odd faces is **always even** in planar graphs

Triangle (odd)	Square (even)	Pentagon (odd)
[0,1,2]	[0,1,2,3]	[0,1,2,3,4]
length = 3 ✓	length = 4 ✗	length = 5 ✓

7. Step 3: Minimum Weight Perfect Matching

Match odd faces in pairs, minimizing total distance in dual graph.

```
# All-pairs shortest paths in dual
dist = dijkstra_all_pairs(dual, odd_faces)

# DP over subsets (n ≤ 18) or greedy (large n)
def exact_mwpm(odd_faces, dist):
    dp[0] = 0
    for mask in range(2^n):
```

```

first = first_unmatched(mask)
for j > first:
    new = mask | (1<<first) | (1<<j)
    dp[new] = min(dp[new], dp[mask] + dist[first][j])
return backtrack(dp)

```

n odd faces	Strategy	Cost
≤ 18	Exact DP ($O(n^2 2^n)$)	Optimal
> 18	Greedy 2-approximation	Near-optimal

8. Step 4: Exclude → Max Cut ✂

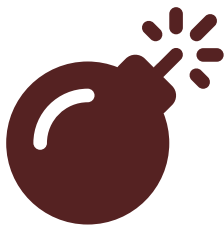
Each matched pair (f_i, f_j) has a shortest path in the dual graph.

| The **primal edges** along that path are **excluded** from the cut.

```

Matched: face 0 ↔ face 1 (distance = 2)
Path in dual: [0, 1]
Edge dual(0,1) → primal(1,3) weight 2
→ Exclude edge (1,3) from cut ✓

```



Syntax error in text
mermaid version 11.14.0

9. Live Code: Python Implementation 🟢

Module: src/netlistx/hadlock.py (60 lines of logic)

```

def solve_hadlock_max_cut(G):
    planar, faces = nx.check_planarity(G)
    # ... 6 steps ...
    return cut_edges # py:set of edges

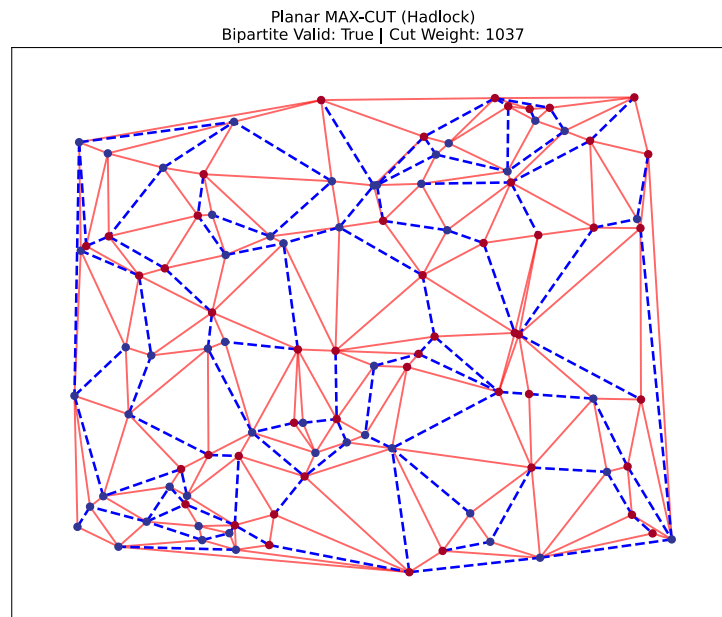
```

Demo: Square with diagonal

```

Input:  edges 5+10+5+10+2 = 32 total
Output: cut 5+10+5+10 = 30 ✓
        excluded = 2 (the diagonal)

```



Hadlock Visualization

10. C++ Port: Performance & Safety 🚀

Header: `include/netlistx/hadlock.hpp` (template, header-only)

Component	Lines	Approach
<code>build_dual</code>	45	<code>std::map</code> edge-face mapping
<code>dijkstra</code>	25	Priority queue
<code>exact_mwpm</code>	45	DP over subsets
<code>greedy_mwpm</code>	20	Greedy 2-approx
<code>solve_hadlock_max_cut</code>	100	Orchestrator
<code>validate_max_cut</code>	30	BFS 2-colouring

C++ Design Decisions: - Templates + callables (not virtual dispatch) 🚀 - `py::set<edge_t>` return type matching Python API - Hash for `std::pair` via specialization - `[[maybe_unused]]` for warning-free MSVC builds

11. Comparison: Python vs C++ VS

Aspect	Python	C++
Face extraction	<code>nx.check_planarity()</code>	Caller-provided

Aspect	Python	C++
Dual graph	NetworkX Graph	<code>vector<vector<DualEdge>></code>
MWPM	<code>nx.algorithms.matching</code>	Custom DP/greedy
Type safety	Duck typing	<code>node_t</code> template param
Weight lookup	<code>G[u][v]['weight']</code>	<code>weight(u, v)</code> callable
Build system	<code>pip</code>	xmake (C++23)









Key C++ advantage: No dependency on NetworkX planar embedding → best for embedded/VLSI tools.

12. Verification

Both Python and C++ pass identical test suite:

```
# Python test
cut = solve_hadlock_max_cut(G)
ok, val = validate_max_cut(G, cut)
assert ok and val == 15

// C++ test (doctest)
auto cut = solve_hadlock_max_cut(G, weight, faces);
auto [ok, val] = validate_max_cut(G, cut, weight);
CHECK(ok);
CHECK_EQ(val, 15);
```

Test Case	Python	C++
Triangle (3-cycle)	 15	 15
Square + Diagonal	 30	 30
Grid (bipartite)	 all edges	 all edges
Empty graph	 empty	 empty

13. Complexity Analysis

Step	Complexity	Notes
Dual construction	$O(F + E)$	Linear in faces + edges
All-pairs shortest path	$O(F \log F + F^2)$	Dijkstra from each odd face

Step	Complexity	Notes
MWPM (exact, $n \leq 18$)	$O(n^2 2^n)$	DP over subsets
MWPM (greedy, $n > 18$)	$O(n^2 \log n)$	Sorting edges
Total	$O(F^3)$ worst	But $F = O(V)$ for planar

Sparsity wins: $|E| \leq 3|V| - 6$ for planar graphs.

14. Real-World Applications 🌐

VLSI Circuit Partitioning 💻

- Planar chip layouts → Max-Cut optimizes wire length
- Our C++ header fits in EDA toolchains

Network Design 🌐

- Load balancing in planar topologies (e.g., road networks)
- Community detection in social graphs

Computer Vision 🖼️

- Image segmentation as graph cut on pixel adjacency
- Planar MRF inference

15. Lessons Learned 🗨️

1. **Planarity is a gift** — it turns NP-complete into polynomial
2. **Duality is powerful** — graph problems often simplify in the dual
3. **Match Python ergonomics in C++** — templates + callables = clean API
4. **Test the edge cases** — empty graphs, single edges, bipartite inputs

Codebase Stats 📦

File	Language	Lines
hadlock.py	Python	60

File	Language	Lines
test_hadlock.py	Python	170
hadlock.hpp	C++20	420
test_hadlock.cpp	C++	200

16. References

- Hadlock, F. O. (1975). "Finding a Maximum Cut of a Planar Graph in Polynomial Time." *SIAM Journal on Computing*, 4(3), 221–225.
- Karp, R. M. (1972). "Reducibility among combinatorial problems." *Complexity of Computer Computations*, 85–103.
- Goemans, M. X. & Williamson, D. P. (1995). "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming." *JACM*, 42(6), 1115–1145.

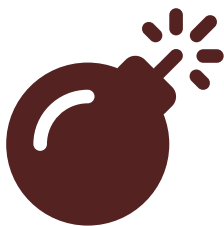
Repos

- Python: github.com/luk036/netlistx → `src/netlistx/hadlock.py`
- C++: github.com/luk036/netlistx-cpp → `include/netlistx/hadlock.hpp`

Thank You!

Questions?

"The maximum cut problem in planar graphs is solvable in polynomial time." — Hadlock, 1975



Syntax error in text
mermaid version 11.14.0