Multilevel Combinatorial Optimization System (Quantum-Classical Hybrid)

# Overview

This document presents a reference design for a multilevel combinatorial optimization system that leverages quantum-classical hybrid architectures. The system is capable of accepting real-world problem inputs, mapping them into standard mathematical models such as QUBO or Ising, and processing them using appropriate classical, quantum, or hybrid solvers to achieve high-quality solutions.

# Text-Based System Architecture

+-----------------------------------------------------------------------------------+  
| USER INTERFACE |  
| |  
| - Problem Submission Portal (Web/API/CLI) |  
| - Use Case Selector: TSP | Graph Partition | Community Detection | Scheduling |  
| - Constraints & Goals Input (e.g., minimize cost, maximize throughput, etc.) |  
+--------------------------------------+--------------------------------------------+  
 |  
 V  
+-----------------------------------------------------------------------------------+  
| USE CASE INTERPRETER MODULE |  
| |  
| - Parses Problem Description |  
| - Maps to Internal Templates (TSP, Knapsack, MaxCut, etc.) |  
| - Suggests suitable encodings and solver types |  
+--------------------------------------+--------------------------------------------+  
 |  
 V  
+-----------------------------------------------------------------------------------+  
| QUBO/ISING ENCODER LIBRARY |  
| |  
| - Converts classical problem → QUBO/Ising models |  
| - Encodes constraints as penalties |  
| - Uses problem graph → adjacency matrix → binary form |  
+--------------------------------------+--------------------------------------------+  
 |  
 V  
+--------------------------------------------------+--------------------------------+  
| MULTILEVEL SOLVER ORCHESTRATOR MODULE | |  
|--------------------------------------------------| |  
| - Solver Selection Engine | |  
| · Chooses Classical / Quantum / Hybrid | |  
| · Based on problem size, sparsity, and depth | |  
| | |  
| - Multilevel Scheduler | |  
| · Stage 1: Classical Approximation | |  
| · Stage 2: Quantum Substructure Solvers | |  
| · Stage 3: Classical Post-processing | |  
+----------------+---------------------------------+--------------------------------+  
 |  
 V

+--------------------------------------------------------------------------+  
| SOLVER LIBRARY AND EXECUTION ENGINE |  
| |  
| +------------------+ +------------------+ +--------------------+ |  
| | Classical Solvers| | Quantum Solvers | | Hybrid Solvers | |  
| |------------------| |------------------| |--------------------| |  
| | Sim. Annealing | | QAOA | | QAOA + RL | |  
| | Genetic Alg. | | VQE | | Classical Init | |  
| | Swarm Opt. | | Grover Search | | Quantum Refinement | |  
| +------------------+ +------------------+ +--------------------+ |  
+-----------------+-------------------+---------------------+--------------+  
 | | |  
 V V V  
+-------------------------------------------------------------+  
| HARDWARE BACKEND EXECUTION MODULE |  
| |  
| - Quantum Devices: IBMQ, D-Wave, IonQ, Rigetti, etc. |  
| - Simulators: qiskit-aer, cirq, forest, etc. |  
| - Classical HPC Clusters for hybrid orchestration |  
+----------------------+--------------------------------------+  
 |  
 V  
+-------------------------------------------------------------+  
| RESULT AGGREGATOR AND INTERPRETER |  
| |  
| - Collates solver results |  
| - Decodes binary/QUBO back to human-readable form |  
| - Applies metrics: energy, convergence, performance |  
| - Visualizations: Graph plots, schedules, clusters |  
+----------------------+--------------------------------------+  
 |  
 V  
+-------------------------------------------------------------+  
| RESULTS & FEEDBACK LAYER |  
| |  
| - Returns results to user |  
| - Suggests alternate solvers if needed |  
| - Allows reruns with changed parameters |  
+-------------------------------------------------------------+

# Sample Python Solver Demo (Max-Cut)

The following code implements a simple multilevel optimization demo using simulated annealing to solve a Max-Cut problem encoded into QUBO format.

Python Code Snippet:

import numpy as np  
import networkx as nx  
import random  
  
def get\_user\_input():  
 G = nx.Graph()  
 edges = [(0,1,2),(0,2,1),(1,2,3),(1,3,2),(2,3,2),(3,4,1),(4,0,3)]  
 G.add\_weighted\_edges\_from(edges)  
 return G  
  
def build\_qubo\_maxcut(G):  
 Q = {}  
 for i, j, w in G.edges(data='weight'):  
 Q[(i,i)] = Q.get((i,i), 0) - w/2  
 Q[(j,j)] = Q.get((j,j), 0) - w/2  
 Q[(i,j)] = Q.get((i,j), 0) + w/2  
 return Q  
  
def simulated\_annealing(Q, num\_reads=10):  
 n = max(max(i,j) for i,j in Q.keys()) + 1  
 best\_sol = None  
 best\_energy = float('inf')  
 for \_ in range(num\_reads):  
 sample = [random.choice([0,1]) for \_ in range(n)]  
 energy = sum(Q.get((i,j),0)\*sample[i]\*sample[j] for i,j in Q)  
 if energy < best\_energy:  
 best\_energy = energy  
 best\_sol = sample  
 return best\_sol, best\_energy  
  
def interpret\_solution(G, solution):  
 return [(u,v) for u,v in G.edges() if solution[u] != solution[v]]  
  
def run\_trial(seed=None):  
 if seed: random.seed(seed); np.random.seed(seed)  
 G = get\_user\_input()  
 Q = build\_qubo\_maxcut(G)  
 sol, energy = simulated\_annealing(Q, 20)  
 cut = interpret\_solution(G, sol)  
 print("Cut:", cut, "Energy:", energy, "Partition:", sol)  
  
run\_trial(seed=42)

# \*Multilevel combinatorial optimization system using a QUBO-based solver. \*

We'll simulate a Max-Cut graph partitioning problem – a classic combinatorial optimization problem often used in quantum computing benchmarks. This is part of the **Multilevel Combinatorial Optimization System across Quantum Architectures**

This example includes:

1)  A simulated user input  
  
2) A basic QUBO encoder  
  
3) A simulated hybrid solver (Simulated Annealing + classical heuristics)  
  
4) Multilevel execution pipeline  
  
5) Re-run capability with different random seeds

**🔁 Multilevel Combinatorial Optimization Demo (Max-Cut)**

🧠 User Input: Max-Cut problem on weighted undirected graph

✅ Solver Result

Cut: [(0, 1), (1, 2), (1, 3)]

Cut size: 3

Energy (QUBO): -7.0

Node partition: [1, 0, 1, 1, 1]

🔁 Rerunning trial with new seed...

🧠 User Input: Max-Cut problem on weighted undirected graph

✅ Solver Result

Cut: [(0, 2), (0, 4), (1, 2), (2, 3), (3, 4)]

Cut size: 5

Energy (QUBO): -7.0

Node partition: [1, 1, 0, 1, 0]

,,