A3 Branch and Bound Partitioning

Summary

In this assignment, Kernighan-Lin algorithm is implemented to do bi-partitioning. A graph and presents node and its connections and a dictionary(HashMap) which stores all the edges are created. We calculate the edge cut size and net cut size when partitioning.

Algorithm

Graph

We create a graph to store nodes and their connections. The graph is presented by a python dictionary, where the key is the node and value is a set which contains all the nodes it directly connecting to.

```
graph = {node0: (node1, node2),
        node1:(node0, node3),
        node2:(node1),
        ...}
```

Edges and Edge cut size

We use python dictionary to store all the edges.

Key is a `frozenset` which contains the two vertices(nodes) of the edge. Value is the status of the edge (whether it's cut). True represents `is cut` and False represents `not cut`. This is used to calculate the edge cut size of the bi-partitioning. Edge cut size stands for the number of edge that crosses partitioning.

Net cut size

We also calculated the net cut size. The net cut size stores the number of net which crosses partition.

Kernighan-Lin / Fiduccia-Matheyses

Kernighan-Lin algorithm is implemented to do bi-partitioning.

We use max heap queue to stores the unlocked nodes so that we can get or pop the nodes with the highest gain.

```
function partition(num_passes):
    initial gains of nodes
    get current cutsize, partition, edges
    min_edge_cutsize = current edge_cutsize
    for _ in range(num_passes):
        unlock all nodes
        while chip has unlocked nodes:
            node = select_node()
            move_node(node)
            cutsize -= node.gain
```

```
            if edge_cutsize < min_edge_cutsize:
                update min_edge_cutsize
                store current partition
                store edges
        rollback_to_saved_partition(partition_copy, edges_copy)
```

When selecting node, we select nodes whose move would not cause an imbalance. In every iteration, we store the partition with min edge cutsize

When moving nodes, we also update gains of all the nodes that connects to the our selected node.

Test

A test file is created for testing the function which calculates the edge cutsize. We load the benchmark file `cm82a.txt`. `block0` and `block1` is created to store the bi-partition results. Nodes with even id are assigned to `block0` and nodes with odd id are assigned to `block1`.

```
expected_result = 0
# all nodes with even id in block0 and all nodes with odd id in block1
for node_id in chip.graph_id:
    for nei_id in chip.graph_id[node_id]:
        if nei_id % 2 != node_id % 2:
            expected_result += 1

print("expected", expected_result)
def test_calc_edge_cutsize():
    assert chip.calc_cutsize() == expected_result
```

Test is passed.

`1 passed in 0.45s`

GUI

We use `tkinker` to select benchmark files and represent final partition result.

Results

benchmark files	edge cutsize	net cutsize
cc	18	10
cm82a	2	1
cm138a	15	9
cm150a	11	7
cm162a	15	10
con1	6	4
twocm	1	1
ugly8	15	8
ugly16	63	16
z4ml	6	5