CPEN513 Final Project - K-Way vs. Bi-Recursive Partitioning

Brandon Freiberger
86799228

I. Description

The goal of my final project was to compare performance in terms of cost (number of edge cuts) and run time between recursive bi-partitioning and k-way partitioning. While neither methods were very optimized, similar data structures were used in both, suggesting that the results may be on the right trend.

II. General Structure

A. Recursive Bi-Partitioning

The recursive bi-partitioning implementation was based on A3. It uses the same data structures:

- edges: Stores the edges, and the nodes associated with each edge. Also stores if the edge is currently cut.
- nodes (x2): Stores the nodes in partition a and b, respectively. This structure also contains the list of edges each node is associated with, the current gain, and if the node is locked or unlocked.

Most of the code was able to be reused, with the main modifications coming from the cost and gain functions. There is an additional file used to recursively call the partition method.

B. K-Way Partitioner

The general design of the K-way partitioner follows the work done by Karypis et. al [1]. At a high level, their algorithm works as follows:

1) Coarsen the graph by grouping together vertices who do not share an edge
2) Once Coarsened, perform K-Way partitioning
3) Uncoarsen the graph

Unfortunately, due to time constraints, coarsening and uncoarsening were not implemented. While this would have made the final project better at reducing edge cuts, it should be noted that the authors used the formula $k * c$ to determine the final number of nodes in each partition, where c was determined to be 15, and k was the number of ways of the partition. Because the provided benchmarks for A2 and A3 all contained under 1500 nodes, their coarsened graph should be reasonably close in size to this implementation.

Swaps are made on a node by node basis. A balancing factor of 1.03 was used [1], meaning that the maximum size of a partition would be 1.03x (nodes / number of partitions). Similarly, a minimal partition size of .9x (nodes / number of partitions) is defined.

To determine the gain of each node, the authors keep track of the different partitions that a given node is attached to, the number of attached nodes in the same partition as the node, and the number of attached nodes in each of the other partitions. Based on this, the node is moved if the degree of the other partitions is higher than the internal degree of the partition. It will also be moved to balance partition sizes if the degree of internal and external partitions is the same.

The K-way partitioner uses the node and edges data structure from the bi-recursive partitioner, while also using the vertex_info data structure. The vertex_info structure holds the partition number the node is in, the partitions that its adjacent nodes belong to, the number of adjacent nodes in each partition, and the lock status of the node. Using this general structure makes it easy to perform swapping nodes between partitions.

C. Efficiency Improvements

Both partitioning methods avoid updating any parameters unless the respective node or edge is impacted by a node move.

In order to have consistency between the 2 algorithms, the same seed was used for both. The outputs are verified to ensure that the nodes in each partition are valid, as well as ensure that the costs are appropriately calculated.

III. Results

The trend that can be seen in Table III is that while the K-Way partitioning tends to be faster, the Bi-recursive partitioning tends to be much better at partitioning in terms of cost. This makes sense, as Bi-Recursion will need to repeat log(n) times. Note that there were issues with pair A/B for bi-recursive partitioning past 2 partitions; thus, they are omitted from the table.

Something else to notice is that K-Way partitioning tends to converge much sooner than bi-recursive partitioning. For example, when running the cm138a benchmark, the k-way partitioning converged within 1-2 iterations, depending on the seed, while the bi-recursive partitioning may take up to 6 iterations to converge to a minima. Again, it is likely that this is due to an issue with the function determining whether or not to move a node to a different partition.

The difference in run time seems slightly greater than reported in the work by Karypis et. al [1], again indicating that some of the performance seen in Table III is due to an error in the gain function computation. The amount of edge cuts should be much closer as well (almost equal).

TABLE I
RUN TIME AND PERFORMANCE OF K-WAY AND BI-RECURSIVE PARTITIONING

Benchmark (# Edges)	# Partitions	Partitioning Method	
		K-Way	Bi-Recurs.
cm138a (16)	2	8 (.00114s)	5 (.00163s)
	4	11 (0.001s)	7 (0.00385s)
	8	12 (0.00112s)	14 (0.0061s)
	16	15 (0.0012s)	15 (0.0058s)
ALU2 (207)	2	106 (.038s)	38 (0.237s)
	4	158 (0.0302s)	53 (0.350s)
	8	173 (0.0455s)	61 (0.457s)
	16	178 (.038s)	67 (.52s)
Pair A (814)	2	277 (0.275s)	65 (4.26s)
	4		
	8		
	16		
CPS (773)	2	377 (0.288s)	143 (3.025s)
	4	562 (0.335s)	183 (4.968s)
	8	627 (0.591s)	209 (5.942s)
	16	657 (0.723s)	244 (6.829s)
Apex4 (1271)	2	595 (0.510s)	294 (7.76s)
	4	877 (0.67s)	425 (12.07s)
	8	990 (1.49s)	500 (14.94s)
	16	1078 (1.64s)	538 (17.56s)
cm162a (32)	2	10 (0.0013s)	6 (0.0064s)
	4	26 (0.0016s)	12 (0.011s)
	8	30 (.00148s)	17 (0.0147s)
	16	32 (.0017s)	22 (0.0226s)

IV. RUNNING THE CODE

There is a README file in the github (https://github.com/Undeadpapaya/CPEN513_Project) that provides a guide on how to run the program.

To get up and running quickly, run the command ”python3 tests.py”. To change the input file, modify line 18 of tests.py. Similarly, the number of partitions can be modified on line 22. Unfortunately, the random seed needs to be manually modified manually on line 20.

There is verification to check that there is no missing or extra nodes in both the k-way partitioning and bi-recursive partitioning. Both the run time and the cost of each method will be displayed in the terminal after being run. On the largest benchmark (Apex4), it takes around 20s to run with an Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz with 16GB of ram.

V. REFERENCES

[1] G. Karypis and V. Kumar, ”Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs,” Supercomputing ’96:Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, 1996, pp. 35-35, doi: 10.1109/SUPERC.1996.183537.