In this report, I simply mention the difference of my work compare to the lecture note and source code. As for those that are not mentioned is assume to be similar to the lecture.

Data structure:

1. Net and Cell:

These two data structures are almost same as provided source code. However, I change getCellList() to getCellListPtr() and getNetList() to getNetListPtr(), in order not to copy the whole list when return, but only return the pointer of the vector instead. Although it might violate the design heuristic (usually we do not want the class private member to be edited by other classes), it can save the time to copy the vector frequently.

2. bucket list:

According to the lecture, it maintains bucket list like a vector, which uses the gain as the entry, and contains a node with corresponding gain. If there are other cells with same gain, it link them after the first cell node as double link list. Also, there is a pointer points to the max gain entry.

However, after some of the cells updating their gains, we need to update the position of nodes. During this process, we might need to consider if we delete the first node of the chain (if so, we need to record the second node as the new header of the chain), and also need to update the max gain pointer. This might waste some time and make the logic become complicate.

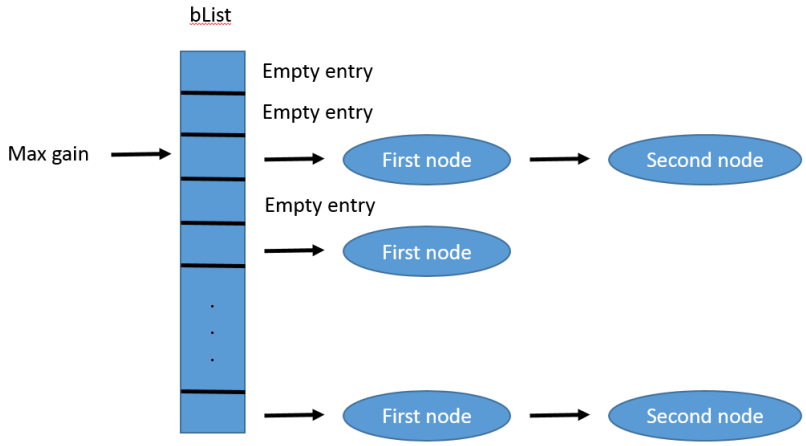
As the result, I do some modification on this data structure by changing it to map container in C++. At the beginning of each round in FM-algo, the bucket list is empty. During calculating the gain for each cell, once we need to open a new entry to insert a node with newly seen gain, we add a dummy node as the header first, and then link the first cell node afterward. By doing so, we do not need to worry about whether we delete the first cell node or not in the future, since there is always a dummy node left as the header of the chain.

In addition, once the node is moved and locked, we delete the node in the bucket list since it is not able to become a candidate. It can reduce the size of recorded node. When we delete the last node of the entry, we delete the dummy node as well and erase the whole entry. I assign the gain value as the Id of the dummy node, so that I can easily find the entry where the dummy node belongs to. The advantage of deleting

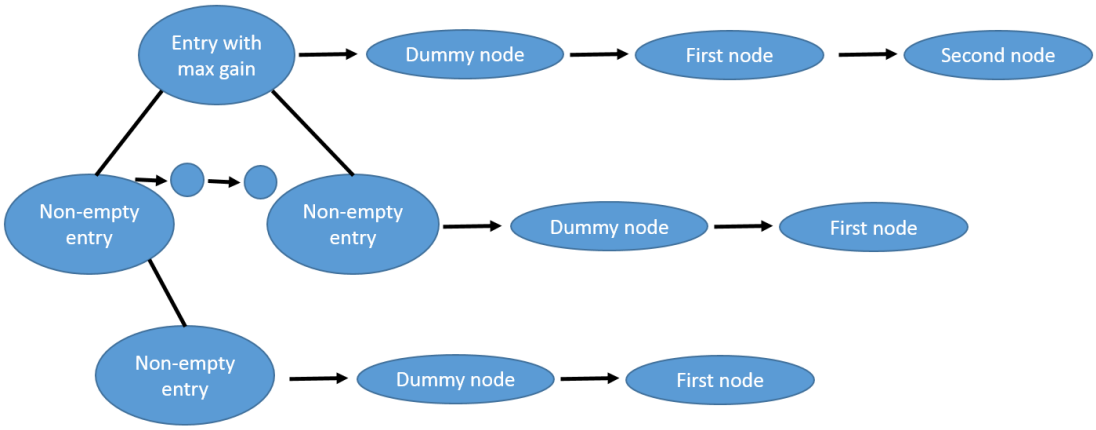
the empty entry is that we no longer need to maintain a pointer points to the max gain entry any more. Since the map container in C++ is a red-black tree, if we guarantee to erase the empty entry, the root of the tree is always the max gain entry by the property of order.

Assume there are k entry in the bucket list. The delete and insert of the entry is only O(log k) in red-black tree, so dynamically maintain the entry of the bucket list is better than updating the max gain pointer by going through the vector, which is O(k).

Original bucket list in lecture:



My bucket list:



Algorithm:

1. initial partition:

A good initial partition might lead to a good local optimal solution. I have tried different heuristic ways, such as going through each net and put all the cells to the same part until two parts are balanced. However, I found that it costs some time and do not guarantee to improve the cut size after some experiment. As the result, I decide to simply separate the cells fifty-fifty according to the index.

	50/50		Heuristic on net (inc)		Heuristic on net (dec)	
	Total time	Cut size	Total time	Cut size	Total time	Cut size
Input 0	2429.42	12214	2333.84	12495	2756.34	12149
Input 1	0.022	1240	0.028	1236	0.026	1223
Input 2	0.056	2204	0.072	2212	0.047	2262
Input 3	5.954	27019	4.337	27101	5.872	27211
Input 4	20.99	44791	17.51	44867	18.06	45630
Input 5	57.06	143110	134.915	142682	113.82	143482

2. reset before a new round

Before we do the next round, we need to reset the state of cells to the iteration where maxAccGain happened. There are several ways to reset the cell states. We can open a big vector to record the whole cell information in each iteration, so that we can immediately reset to a specific timestamp. However, it cost large memory and copy needs time. Another way is to undo the movement in each iterations one-by-one. However, in the previous section we mentioned that the node is deleted from the bucket list once it is moved and locked, so it is not easy to undo the action in my implementation. Finally, I choose to simply record the index of the moved cell in each iteration, and undo the movement alone. Afterward, we treat it as a new initial partition and re-calculate the gain, net part count, cut size, etc.