Assignment 2 - Placement
EECE 583
Ken Mansfield
February 8, 2016

Info:

Developed in Linux.
Usage: ./Assignment2.exe [filename] [display mode] [anneal speed]
Display mode
0 = normal
1 = slower animations
2 = Final result only
3 = no graphics, console only.

Anneal Speed:
0 = slow anneal
1 = fast anneal

Ex. ./Assignment2.exe alu2.txt 0 0
Makefile and test script provided.

1. Simulated Annealing:

After reading the netfile and placing the cells on the board, the function ScheduleAnneal() is called. This function constructs the annealing schedule. The number of iterations per temperature is determined by (# of cells)^4/3. Initial temperature is determined by taking the standard deviation of 50 random placements and multiplying it by 20 (20 * STD). Next the annealing is started within a while loop.

After each iteration the Temperature is updated by multiplying it by a constant that is 1.0 (Temperature = Temperature * beta). Through testing, I decided upon a few different values for Beta. When the acceptance rate is around the 0.44 range (determined by checking if it is within the range of 0.3 to 0.5 to allow for some fluctuations between individual iterations) I use the value of Beta = 0.997. Outside of that range of acceptance rates Beta = 0.991 is used. Using a larger value allows more iterations to occur when the acceptance rate is near the ideal range of 44%. To try and keep the acceptance rate around 44%, the window is also shrunk when the rate drops below 44%. Having a smaller window allows swaps to happen with cells that are closer by, which increases the chance that the swap will be accepted.

I initially tried to update the Temperature based on the overall STD on every iteration. Doing this would require storing an array of all costs and then calculating STD for every temperature iteration which would be an expensive operation which would increase the overall time.
The annealing ends when the Acceptance rate = 0%, meaning there was a single Temperature level in which no swaps were accepted. At that point two final Anneals are performed at 0 temperature and the loop ends.

Through many tests, it is clear that performing a slower/longer anneal with more iterations results in lower overall cost. To exemplify this, I added another option = Fast Anneal. For Fast anneal I used beta values of: Beta = 0.985 (if acceptance rate is < 0.5 && > 0.3) else 0.94. This results in an anneal that is about 9X faster than the standard mode, but the cost is only around 1.5% higher.

2. Cost Function

This is a pretty simple calculation. Given a net, I simply grab the smallest and largest X and Y values contained within the net. So deltaX = maxX – miny and deltaY = maxY – minY. The cost is then deltaX + deltaY, which is equal to the half perimeter of the cells contained within the net. The cost function takes into account the height of the routing in between rows. So the Y distance between a Cell in the first row and a cell in the 2nd row would 2 and not 1 because of the routing in between. Hopefully that is the intended way to compute the cost for this assignment.

For each individual swap, however, cells are being swapped, not nets. So to determine whether to take the swap, we need to calculate the cost of all the nets that may have changed. The function CostOfCell(x) finds all the nets that cell X belongs to and adds their individual costs together.

3. Efficiency

The DoAnneal() function is kept as simple as possible so that it can be performed quickly. After choosing two cells and random, it determines the cost before the swap, and the cost after the swap. If the cost has decreased, we use the annealing formula e^(-1*costDelta/Temperature) to determine if we keep the change anyways. If not, we swap it back to its original position. Swapping the cells is on O(1) operation since it is accessed by directly indexing into an vector.

The board is stored as a 2D array sBoard[x][y] with its values either -1 or the number of the cell it contains, thus selecting two random cells is an O(1) operation.

The cells are saved in a vector of cell structs: sCells, which for each cell contains a list of all nets it belongs to. This way calculating the cost of each cell swap is quick since we already have a list of all nets that it belongs to and we do not have to search through the netlist to calculate this cost.

4. Results

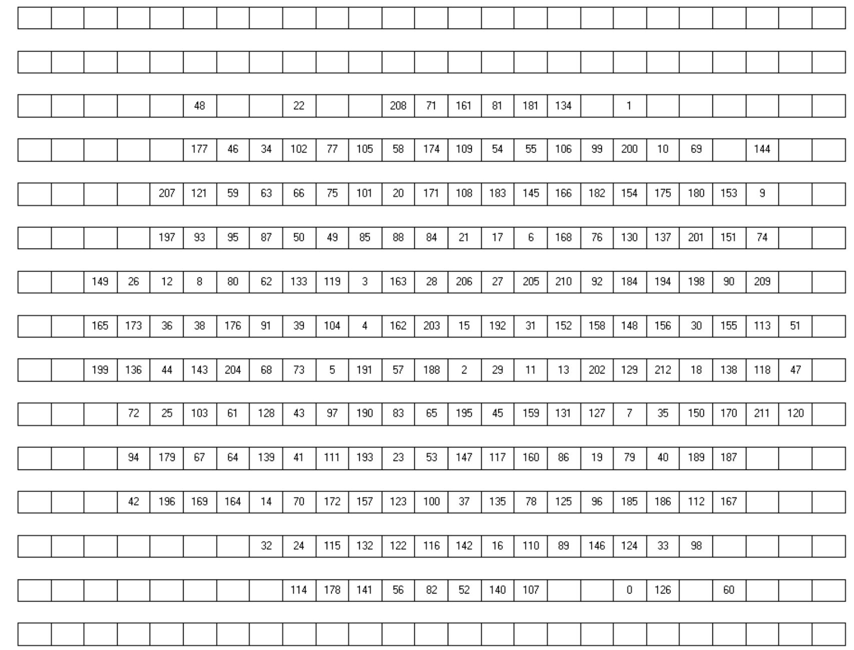
Some sample results. It is evident that increasing the annealing time only improves the quality of the placement by a small amount. The fast Anneal has around 8X less iterations, but the final cost is only between 1-2% worse. View ResultsFastAnneal.nfo and ResultsSlowAnneal.nfo for full results (the output of the testAllFiles.sh).

		Slow Anneal		**Fast Anneal**	
	Initial Cost	**Final Cost**	**T Iterations**	**Final Cost**	**T Iterations**
Alu2	6637	2417	1482	2445	229
Apex1	41264	16016	1520	16516	236
Apex4	83552	32559	1566	33086	242
C880	7200	2876	1468	2931	228
Cm138a	142	72	1293	74	198
Cm150a	328	162	1331	165	200
Cm151a	227	75	2691	76	505

Cm162a	397	161	1357	163	208

5. Graphics

I considered showing individual swaps but the number of swaps is in the order of millions so this would have been pointless. Also, drawing lines for each net would become unruly, overlapping all the numbers. So for this assignment, the graphics are more coarse grained. The entire circuit is drawn for each iteration of T so we can see the placement improve after each iteration. We quickly notice that the cells gravitate towards the middle after the acceptance rate drops below 50%. The acceptance rate, and current Temperature are also shown in the window. The overall results are displayed in the terminal.



6. Initiative

I have given the user extra flexibility to operate the code using different options. Display mode = 0 is the normal operating mode which has a graphics display that quickly updates. Display mode = 1 adds a slight delay between each display update. Display mode = 2 displays the final result only which is useful for testing since displaying the graphics slows down the program. Display mode = 3 disables graphics altogether which is the most useful for testing when we are only interested in seeing the final costs.

I have also given the user the ability to select either a slow anneal or a fast anneal. The fast anneal decreases the temperature faster allowing the algorithm to happen almost 10X faster but with very little drop in quality.

To automate testing, I have provided a script test.sh which executes the annealing program in display only mode on every netfile (*.txt in this case) so that we can quickly test that it works for every file, and so we can examine all the final results.