

ECE 1387 - CAD for Digital Circuit Synthesis and Layout  
**Assignment #2 – Analytical Placement with Heterogeneous Cell Types,  
Spreading, Snapping/Legalizing to Grid**

Austin Liolli  
1005641986

Program Flow

My program consists of 2 main algorithms, which are separated by the initial position and the recursive calls. All data pertaining to block location and net containment is stored in vector format. First, the file is parsed and stored as two separate 2-dimensional vectors, one vector stores the block numbers and the nets it is attached to, the other stores the fixed block numbers and their positions. The program begins by establishing a netlist, which each row of a 2D vector represents the net, and the contents in that row are the blocks attached to that net. This list of nets is used to create the clique weights, the number of vertices in the clique is used in the diagonal matrix definition.

Once all of these vectors are obtained, the Q matrix is solved. The size of the matrix is the difference between the size of the fixedblock vector and the block number vector. This will solve all the non-fixed block locations. Each positions value is calculated by its location, either on a diagonal of the matrix or non-diagonal elements of the matrix. If the position is on the matrix diagonal, the sum of the weights attached to that node are needed. The netlist file is checked by storing a vector of all nets attached to that block. Each net attached is summed for total weight on the node (clique model) and the position is added. If the position is not on the diagonal, then the sum of the weights between that node and its respective position are compared. Two vectors are obtained, one for the block we are in and one for the compare block, if any nets found in these vectors match, one of those weights is added. Next, the B matrix for both X and Y are solved. Similar to the diagonal matrix position, the nets between any fixed blocks and that respective position are checked using two comparative vectors. If there are connections found, that edge weight is multiplied by the respective position the fixed block is at and placed in the matrix.

The next function is to properly calculate the arrays needed to pass to the UMFPACK. This is done by creating 3 arrays which store the position of each first row value, the position that row value is found and the first column position. These arrays are then passed to the linear solver with the result of the x positions and y positions. Using these positions, the first question is completed and the half perimeter wire length is computed.

The second question is executed very similar to the first, and same with the recursion. After the initial split, the above algorithm is repeated for each block set to solve the updated positions. To determine which blocks will be pulled toward each quadrant, the median position in each x and y are determined from that block set (all blocks on first iteration). The blocks are separated by the section which they are closest to. The central position for each split is calculated and a similar algorithm, with smaller vectors and matrices is ran to now compute the updated positions. The highest level positions are updated once this algorithm is ran.

This section is repeated recursively by creating a queue of vectors from the previous iteration. By keeping track of which blocks were split into which quadrant of each iteration along with their respective centers, the split algorithm, which does not take fixed input, can be run again. Depending on the iteration, ie. first split has 4 vectors, second split has 16, the appropriate amount of vectors are popped off the queue to use. As the new vectors are computed they are pushed to the back of the queue to be later used. At the end of each iteration the positions are updated, showing the recursive split for each section.

For normal test cases, the splitting is limited to a position where the solved central position of the next iteration will not be within the first column of blocks. This means that splitting will not occur in very small regions where multiple blocks can be placed in a single location. This is determined by the column size / (4\*iteration). At this point the program will cease to allow cuts and the design can be snapped into place.

Due to time constraints only the first snapping technique was implemented. This is done by ignoring the fixed blocks, as they are already in position, and iterating through the remaining unfixed blocks. Each position of the grid is compared to all of the nodes, and the minimum distance needed to travel from the center of the grid location to that node is the snap node needed. In the positions vector this value is now multiplied by -1, and only positions with positive locations are examined. At the completion of the algorithm, each block shall be located in a specific position for the grid. An issue with the created algorithm for the 4th test case is that when the spreading is run past 4 iterations the snapping does not recognize 4 nodes. For this reason this specific case is limited to this many iterations whereas the other two cases are not limited. With more time to complete, a debug session of where the missing position snaps fell would be found. For the purpose of completion this is limited.

The final output of the program shows the resulting positions of each iteration. The graphics will load with the first question of the respective file selected. (click proceed on the right hand side to initialize after the window pops up). For each case, the iterations are limited. By clicking on the "Spread" button, the next iteration of spreading can be displayed. Once the button is no longer active and the stopping criteria has been met, the snap button will lock the position of the nodes. The snapping can be done at any point in the program after the first iteration. Returned to the console is the HPBBWL of each iteration. At any point, the iteration being viewed can be viewed with or without nets by toggling the "Nets: On" button on the side.

## Results

### Question 1:

File	Half-Perimeter BBWL
cct1	99.8298
cct2	707.656
cct3	2536.62
cct4	7342.21

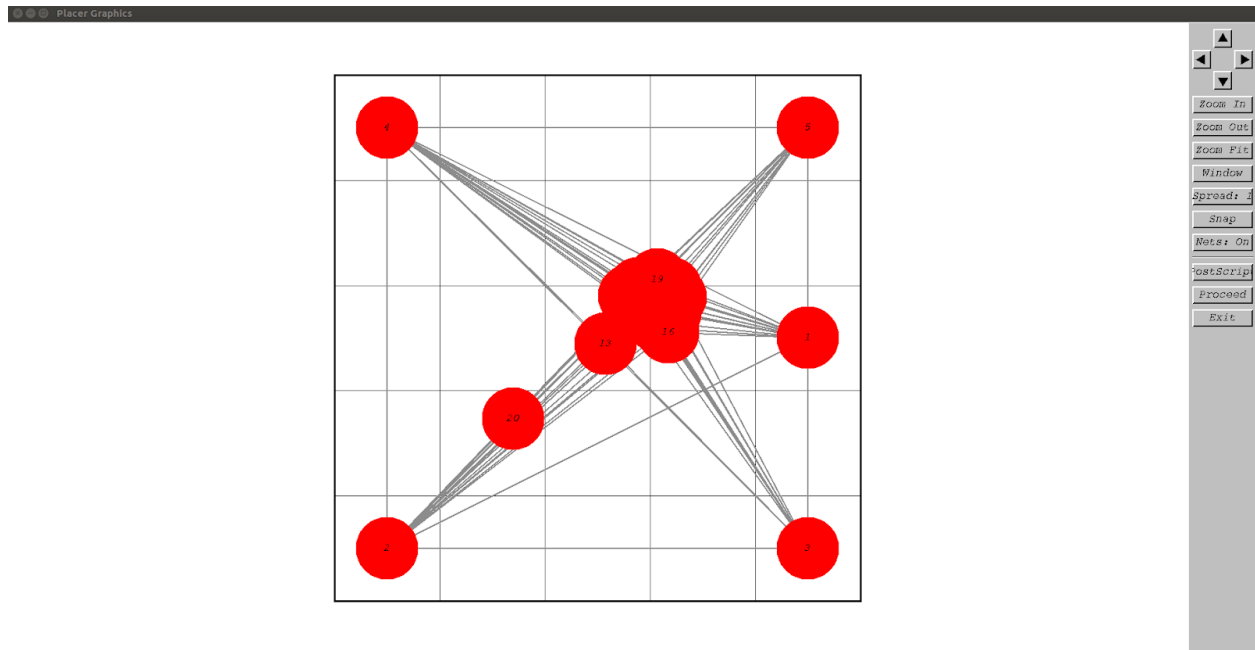


Figure 1: cct1 Initial Placement

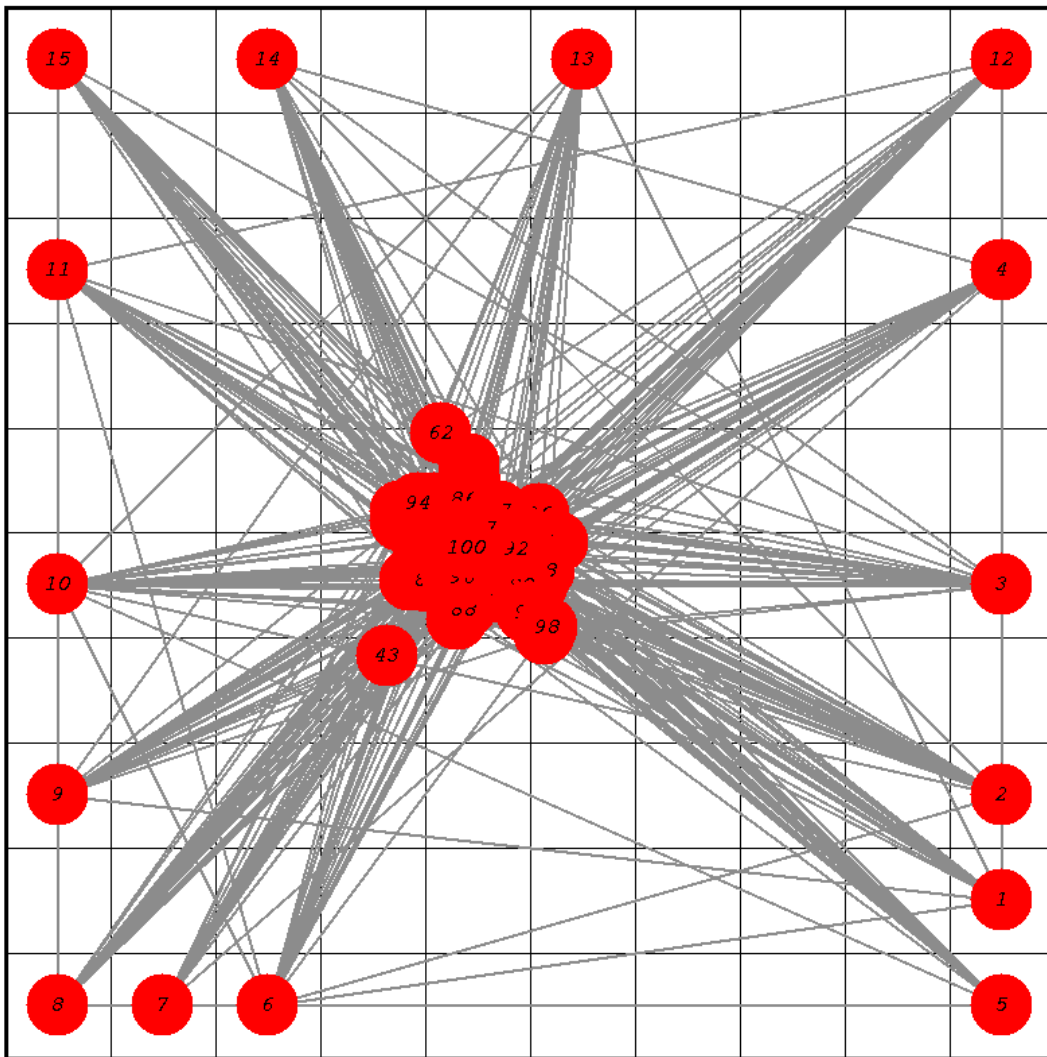


Figure 2: cct2 Initial Placement

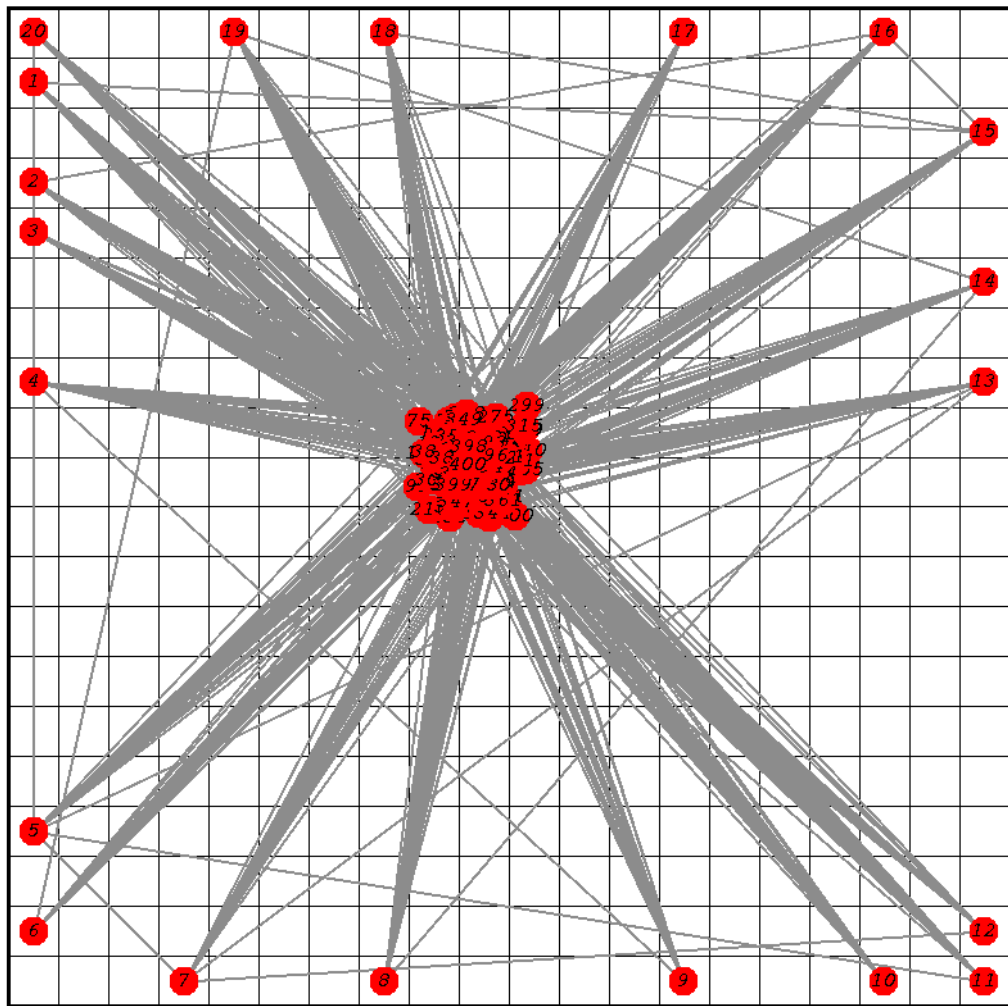


Figure 3: cct3 Initial Placement

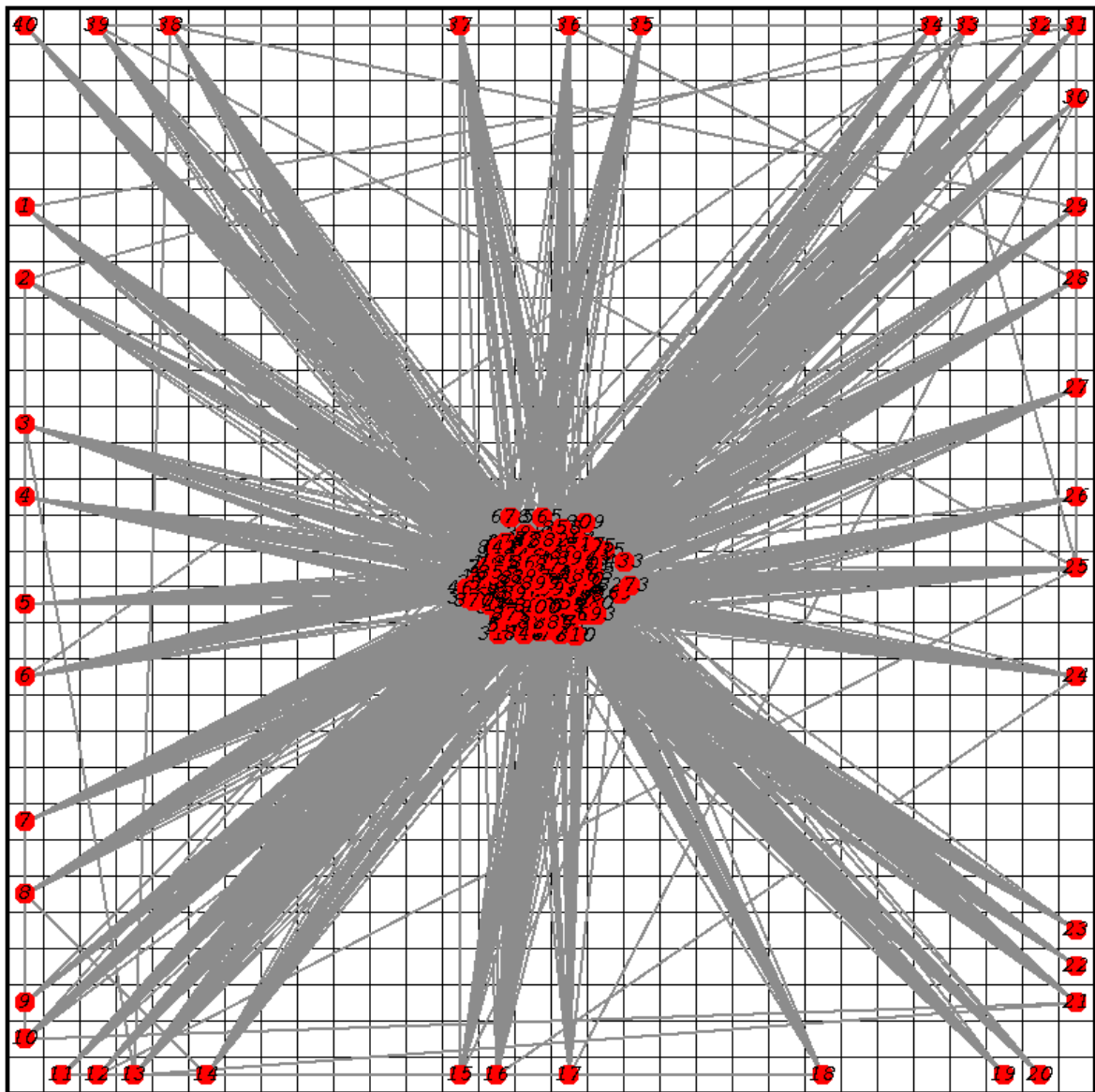
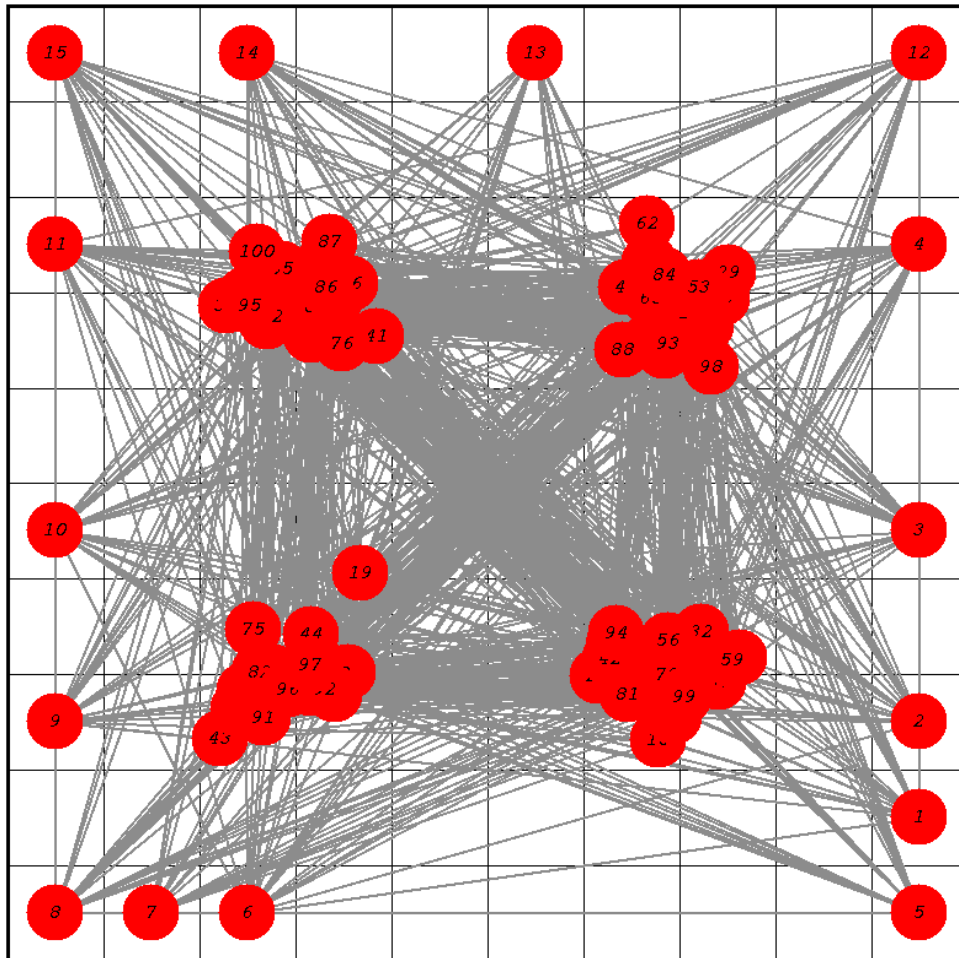


Figure 4: cct4 Initial Placement

*Question 2:*

With each new fixed block added, the wire length is increasing. This is because the initial position solved in question 1 is most optimal, regardless of actual location on the grid. As these fixed blocks are added, the nodes are pulled further from their optimal positions, meaning that with a larger weight, the farther from optimal they are the larger the wire length will be.

File	W=0.5	W=1	W=2	W=5	W=10	W=20
cct2	937.16	981.1	1080.38	1240.07	1330.33	1392.93
cct3	5493.37	6597.3	7703.63	8716.85	9108.05	9263.61
cct4	17050.7	20690.1	24164.4	27248.5	28401.3	28921.2



*Figure 5: cct2 First Spread (W=10)*

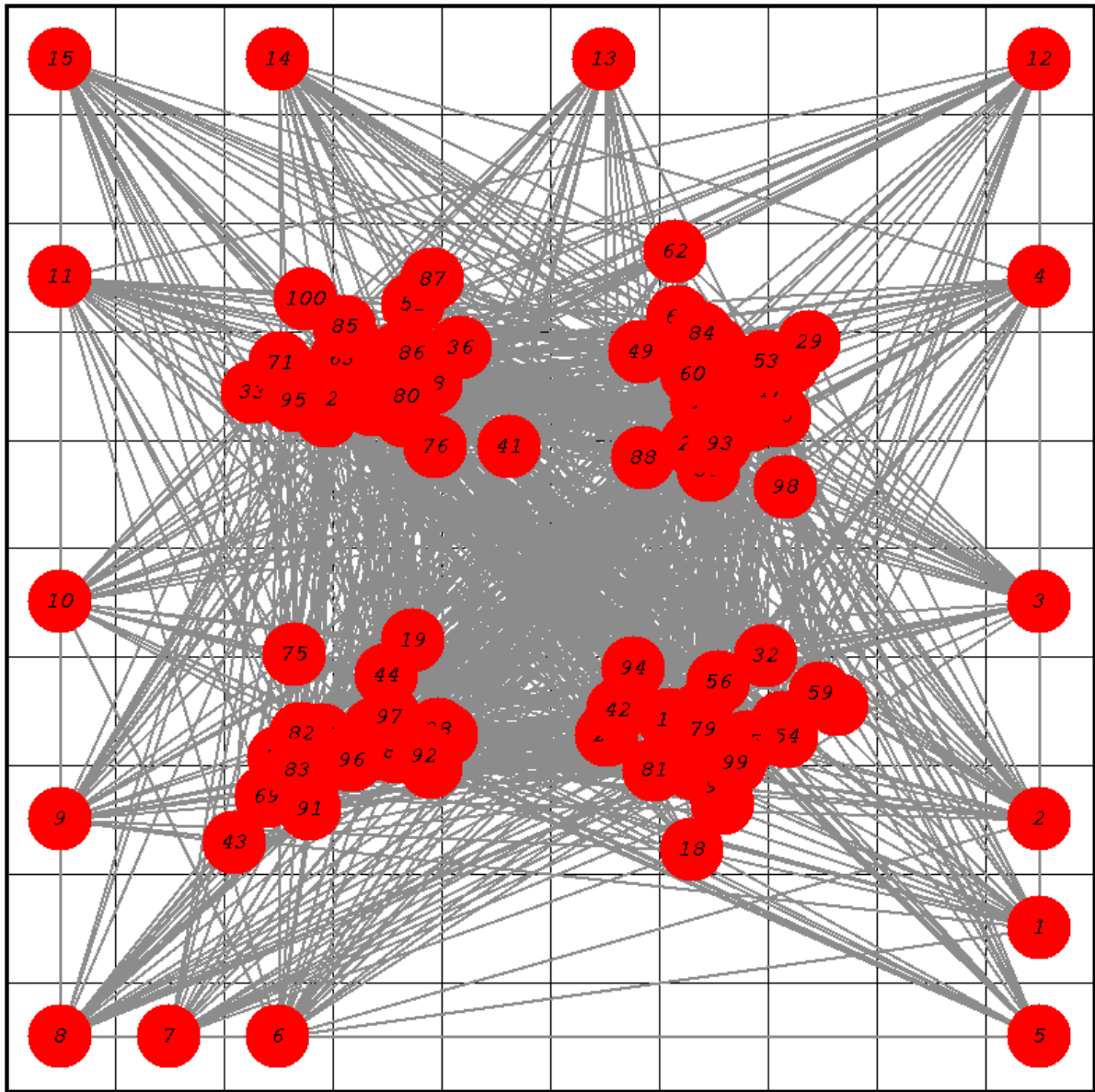


Figure 6: *cct2* First Spread ( $W=5$ )



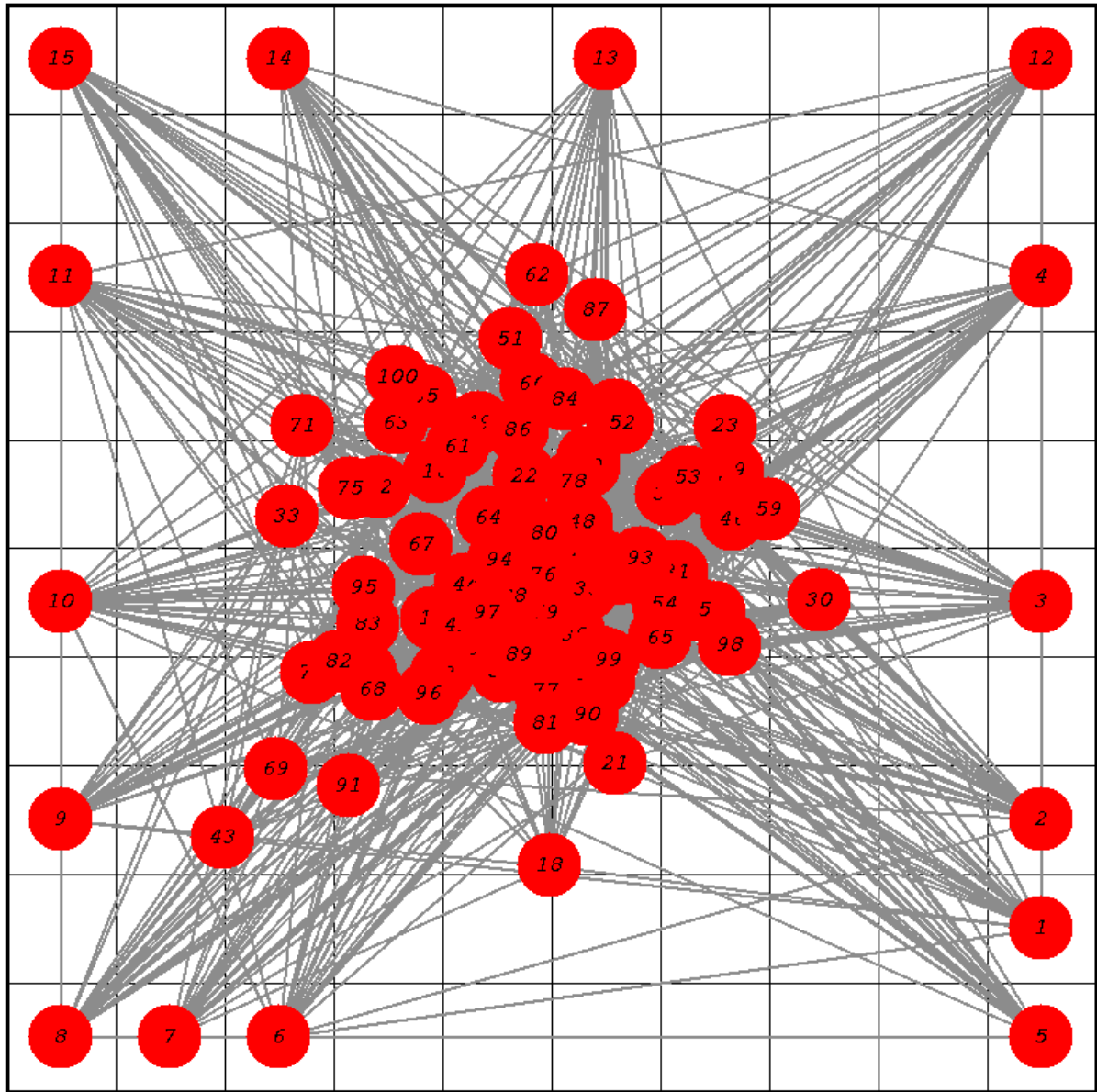


Figure 7: cct2 First Spread ( $W=1$ )

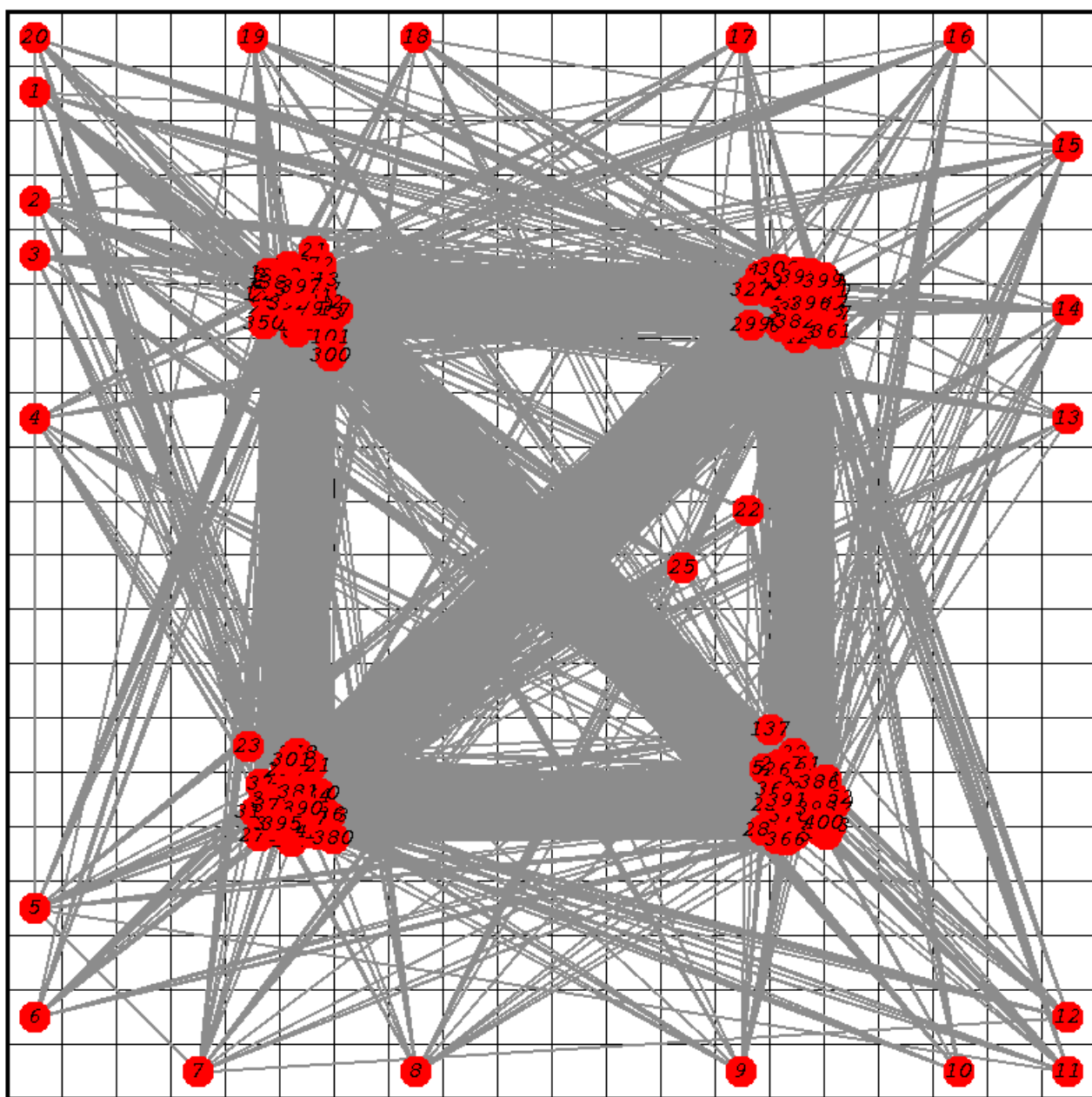


Figure 8: cct3 First Spread

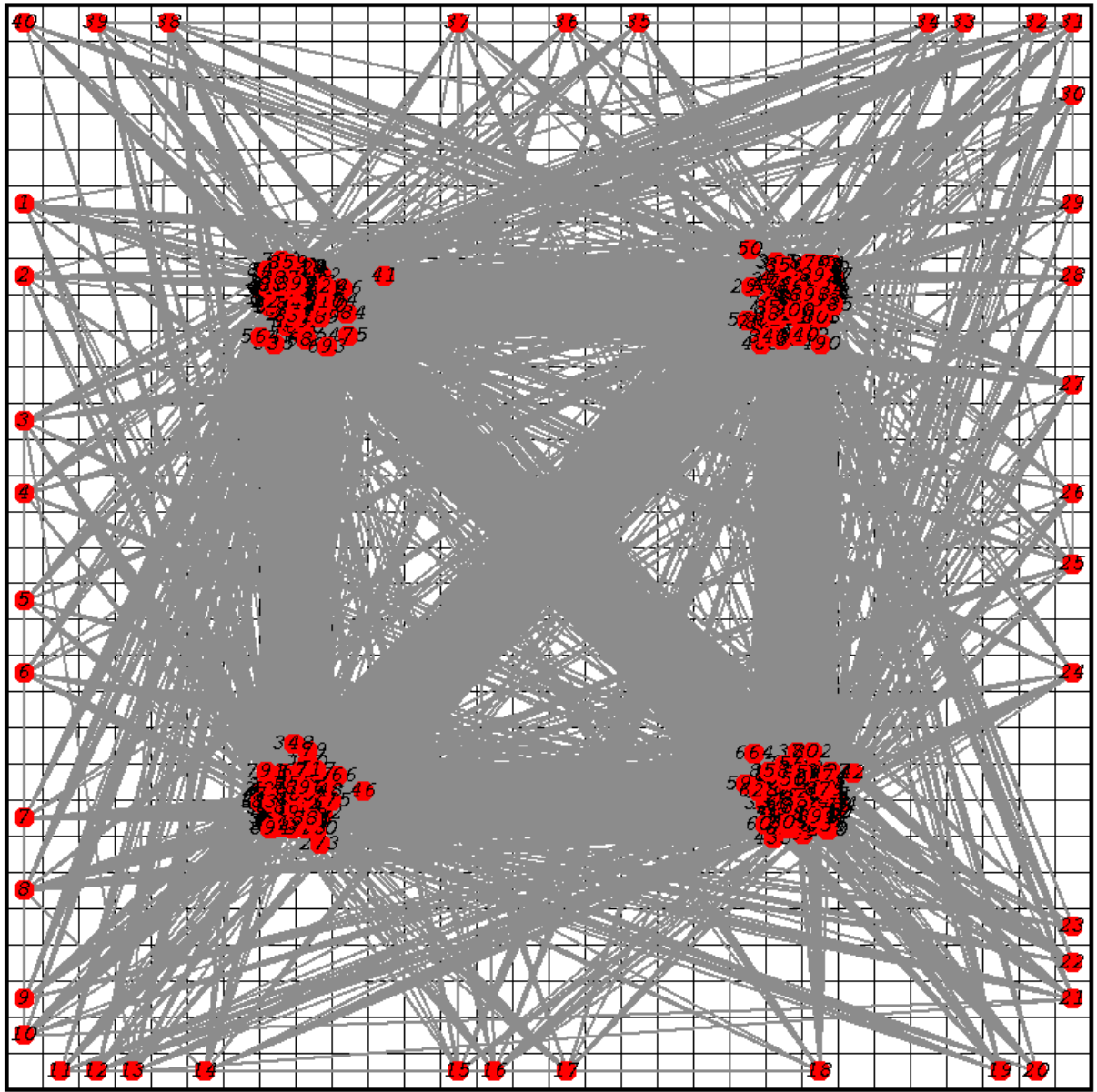


Figure 9: cct4 First Spread

Question 3:

File	HPBBWL, W=20
cct2	1538.36
cct3	11636.3
cct4	36947

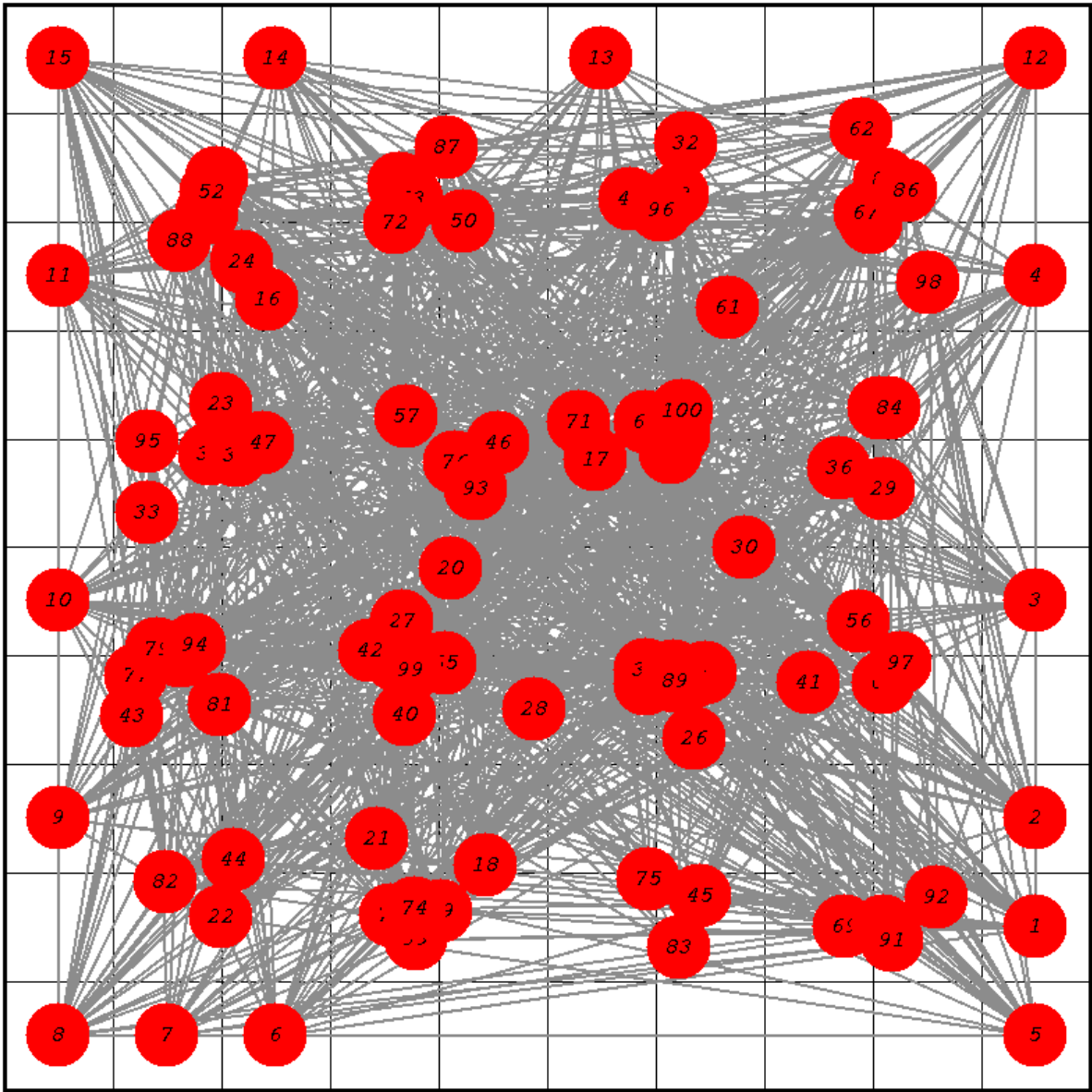


Figure 10: cct2 Full Spread

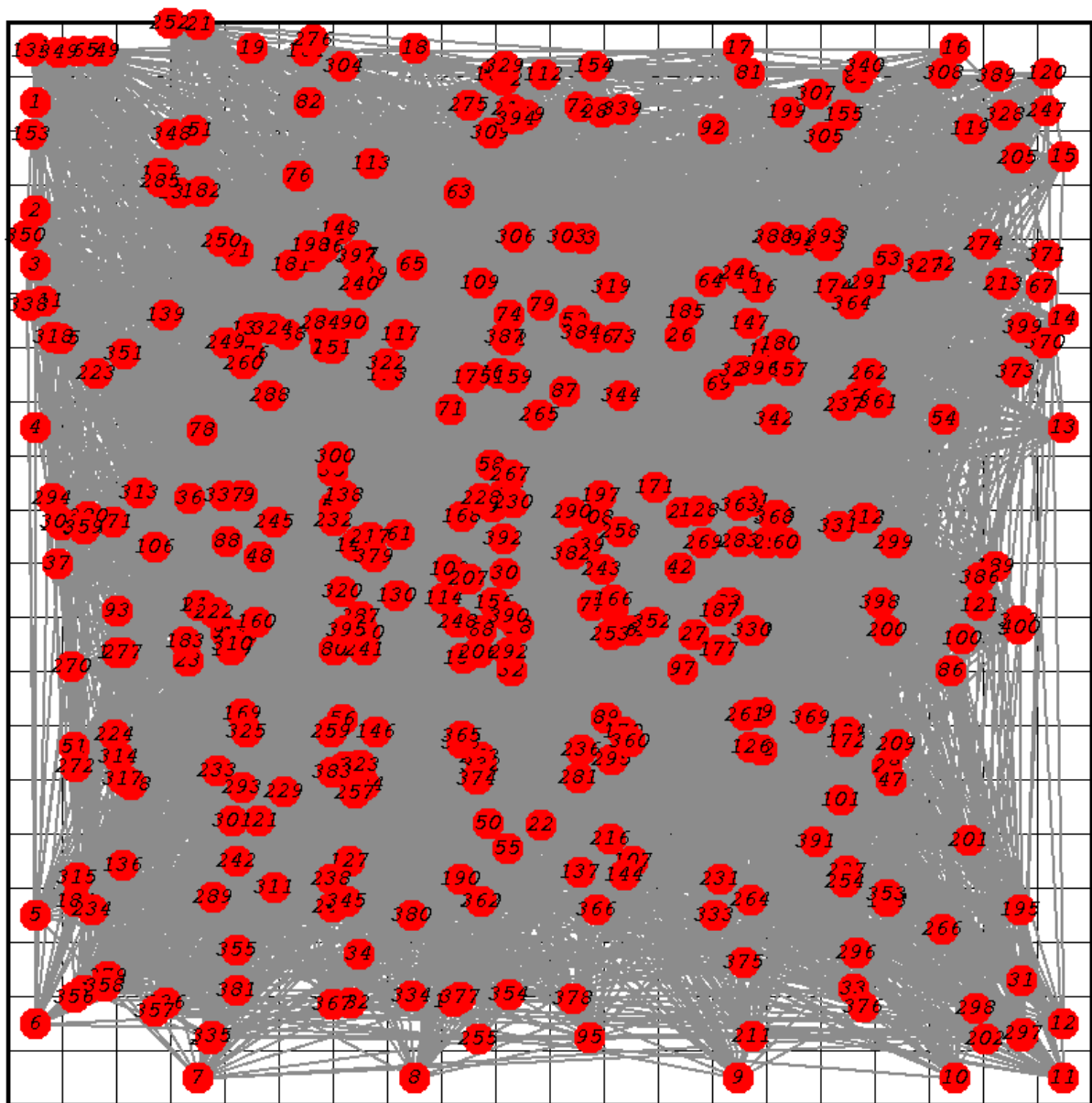


Figure 11: cct3 Full Spread



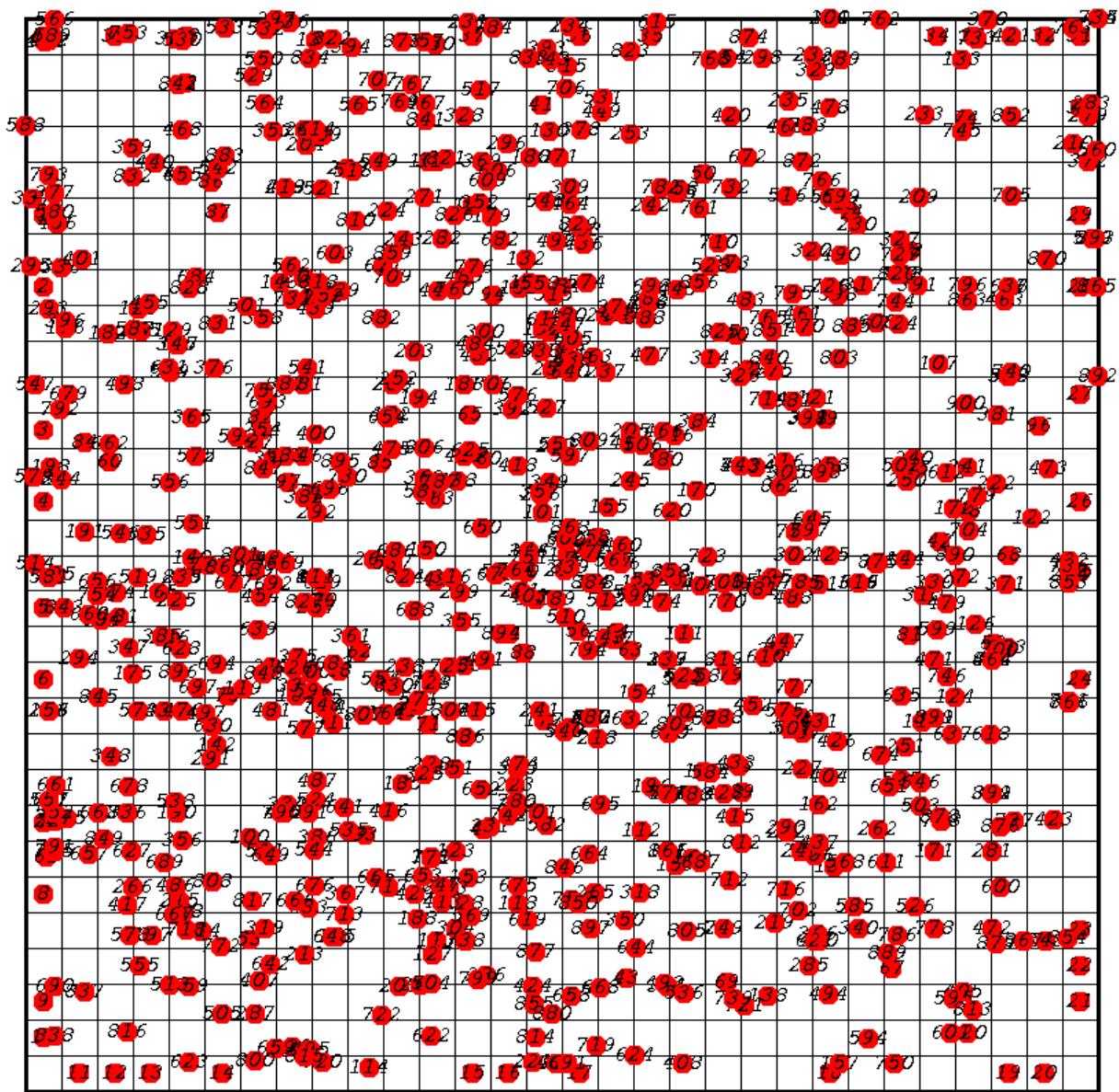


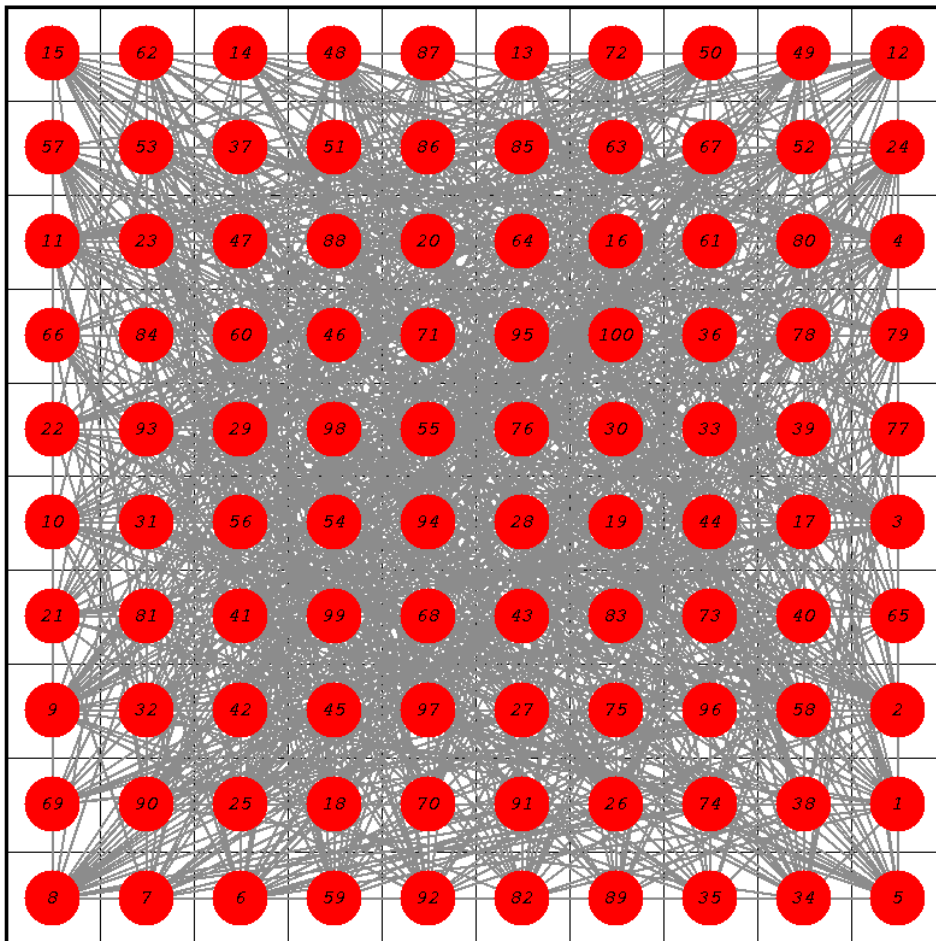
Figure 12: cct4 Full Spread (shown without nets)

Note: this spread level is not implemented in the program for bug where last 4 nodes of grid are not recognized. In final implementation cct4 is limited to 4 spreads.

**Question 4:**

The increase between the wirelength from the last iteration of spreading to the snapped locations is less drastic than the difference between the optimal and spread locations. This is because as the spreading increases, the HPBBWL becomes more optimal relative to the final position. If the original design was snapped, the snapped value would be larger than the final solved positions, because incrementally each position becomes slightly more optimal with each iteration.

File	HPBBWL, W=20
cct2	1659
cct3	12580
cct4	39377



**Figure 13: cct2 Snap Locations**

20	151	104	62	19	153	129	18	165	111	389	190	108	17	194	193	106	16	279	57
1	164	282	277	151	110	90	252	129	112	107	286	184	174	21	120	111	149	107	224
179	113	178	182	105	106	81	187	188	281	76	41	186	65	192	140	128	93	67	15
2	84	148	45	132	91	177	285	150	123	53	24	250	267	118	72	163	294	276	146
3	181	280	183	197	80	214	235	26	122	178	246	138	69	208	143	164	128	271	194
284	113	180	163	186	122	198	173	52	126	199	170	79	108	188	225	103	140	142	14
192	196	269	105	131	142	121	157	49	144	153	29	74	176	222	125	149	159	134	31
4	54	66	69	46	100	87	71	205	38	75	23	109	288	137	185	73	159	171	13
126	25	209	162	207	117	152	103	228	149	290	36	197	191	147	275	95	203	138	177
260	115	96	134	213	116	40	245	64	219	133	109	239	131	157	299	62	173	34	278
130	110	42	166	215	247	92	243	83	65	191	121	185	127	198	30	27	112	77	165
183	283	253	123	154	130	292	179	68	195	139	400	55	199	295	150	236	291	43	141
101	261	44	180	70	37	56	168	169	125	58	86	88	196	139	133	32	189	160	158
158	220	147	114	195	171	51	272	172	94	144	226	204	143	145	206	162	154	193	78
22	293	270	244	141	97	99	229	221	296	148	155	104	216	50	174	117	240	248	127
175	181	151	48	230	160	273	146	63	39	232	190	47	131	254	115	124	298	118	287
5	241	145	134	233	116	210	237	98	59	166	262	289	135	102	172	152	161	184	114
102	136	120	259	242	156	155	218	169	201	132	176	274	265	170	297	156	168	182	123
6	258	101	100	212	257	95	227	202	217	137	211	167	238	200	268	119	161	187	12
135	119	256	7	255	263	136	8	60	264	167	266	33	9	61	175	124	10	28	11

Figure 14: cct3 Snap locations (shown without nets)



40	44	39	52	38	63	58	66	14	65	69	17	62	37	50	97	36	70	35	93	72	15	82	39	25	43	34	33	56	32	31				
1	08	41	27	14	21	74	46	92	69	20	631	052	323	492	347	343	167	333	592	443	153	786	391	343	097	358	243	891	355	416	177	55		
9	44	507	468	34	61	58	01	443	922	985	492	377	543	142	453	608	103	331	338	237	067	531	215	603	774	403	938	743	406	84	30			
7	678	427	477	892	357	077	695	172	33	50	5641	063	717	685	54	41	31	747	033	73	296	827	456	136	853	418	102	247	093	118	04			
6	0	62	44	18	02	595	503	461	742	70	60	44	151	673	28	691	927	445	531	042	096	996	553	633	883	524	463	596	881	644	515	61		
1	2	537	526	09	73	14	36	63	78	81	83	367	662	792	891	522	314	492	831	315	597	583	695	157	824	395	873	584	894	175	05	29		
6	01	204	352	938	322	56	59	26	88	835	21	94	52	884	87	613	138	522	005	635	334	612	96	86	41	88	73	98	27	67	881	465	798	98
2	4	562	433	663	453	274	801	734	534	041	671	588	211	325	073	842	253	746	724	996	372	048	294	677	926	061	151	843	61	28				
3	25	525	625	395	742	953	394	013	694	823	763	126	317	055	084	030	88	658	205	582	583	916	511	036	713	83	76	20	26	934	26			
6	11	544	908	625	585	784	553	245	011	913	543	967	656	122	47	85	20	373	122	76	081	639	004	277	574	484	345	762	16	88	497			
6	66	624	001	42	49	52	33	988	316	965	867	763	518	034	038	51	72	45	26	052	551	514	081	61	68	60	313	01	867	844	986	73	27	
3	3	35	90	55	88	53	651	947	818	263	814	753	266	878	683	227	604	635	303	982	788	926	503	486	985	476	698	282	264	716	97			
6	4	51	43	22	822	644	455	831	408	908	017	964	252	426	452	867	045	924	731	992	671	10	83	60	28	823	875	374	023	956	805	46		
4	74	08	063	444	966	25	65	25	23	062	22	54	54	574	363	620	823	041	11	78	93	64	867	94	601	491	015	148	955	722	461	22	26	
81	1	576	928	728	443	574	791	193	635	044	394	662	146	295	708	577	141	142	753	338	398	477	877	513	021	418	356	415	988	64				
45	6	602	921	28	58	4841	721	897	737	298	874	888	094	157	265	975	208	177	235	273	505	343	477	366	101	167	851	252	38	25				
5	18	21	241	951	76	84	63	03	96	82	1798	707	591	684	573	854	648	584	476	865	028	667	152	575	6774	11	932	774	098	971	50			
107	42	67	06	044	365	516	642	803	733	75	99	155	294	742	546	267	308	022	055	427	974	228	75	79	77	72	284	122	486	588	45			
6	3	257	482	946	612	237	64	63	69	453	128	453	568	31	653	037	943	532	112	728	497	005	84	95	14	720	11	115	006	524	76	24		
8	61	535	71	87	84	33	554	167	833	344	051	962	905	093	823	185	558	275	226	187	101	85	48	54	057	530	72	173	391	453	306	07		
8	34	332	655	774	541	871	901	755	388	11	69	48	182	51	628	93	75	3794	282	995	908	426	464	43	77	70	88	907	272	156	161	36		
71	51	86	897	907	701	593	67	46	58	157	388	48	885	241	297	241	814	628	998	643	308	616	328	078	91	66	774	64	93	231	83	56		
7	1	607	91	64	40	72	514	865	996	205	123	325	578	677	426	357	377	037	988	535	93	91	64	42	611	775	258	194	145	917	381	02		
7	17	67	67	66	74	89	66	61	708	501	98	96	585	47	97	66	530	567	587	74	236	957	398	563	206	001	268	12	70	71	37	227	114	85
8	7	568	463	002	865	267	754	775	694	656	115	036	688	717	025	962	292	197	124	926	217	216	285	958	088	556	472	408	134	91				
8	91	274	296	432	688	162	502	126	781	137	162	627	955	432	878	766	224	108	624	586	344	872	134	133	432	186	337	018	36	23				
3	88	946	914	954	062	638	002	208	547	505	682	856	247	863	042	067	991	37	57	51	371	81	081	697	282	074	444	241	388	31	22			
9	6	598	375	192	731	886	907	808	387	72	43	32	128	145	947	860	180	719	24	124	930	12	397	494	376	423	197	254	941	17	21			
10	6	585	112	214	307	718	151	391	125	825	106	274	931	481	008	805	941	188	971	236	534	726	776	675	447	201	097	798	792	60				
197	1	12	13	65	14	80	88	143	52	74	53	881	15	16	623	17	878	861	561	717	784	31	16	880	916	19	92	19	20	52				

Figure 15: cct4 Snap Locations