[4] T. N. Bui, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms", in *Proc. ACM/IEEE Design Automation Conf.*, 1989, pp. 775-778.

[5] P. K. Chan, M. D. F. Schlag and J. Zien, "Spectral K-Way Ratio Cut Partitioning and Clustering", *Proc. Symp. on Integrated Systems*, Seattle, March 1993.

[6] J. Cong and M. Smith "A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design" *Proc. ACM/IEEE Design Automation Conf.* 1993, pp. 755-760.

[7] J. Cong, L. Hagen and A. B. Kahng, "Random Walks for Circuit Clustering", *Proc. 4th IEEE Intl. ASIC Conf.*, Rochester, September 1991, pp. 14.2.1 - 14.2.4.

[8] C.M Fiduccia and R.M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *Proc. ACM/IEEE Design Automation Conf.*, June 1982, pp. 175-181.

[9] T. Fujii, H. Horikawa, T. Kikuno, and N. Yoshida, "A Heuristic Algorithm for Gate Assignment in One-Dimensional Array Approach", *IEEE Trans. on CAD* 6(2), March 1987, pp. 159-164.

[10] J. Garbers, H. J. Promel and A. Steger, "Finding Clusters in VLSI Circuits" *Proc. IEEE Intl. Conf. on Computer-Aided Design*, Santa Clara, Nov. 1990, pp. 520-523.

[11] L. Hagen and A. B. Kahng, "A New Approach to Effective Circuit Clustering", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, Santa Clara, Nov. 1992, pp. 422-427.

[12] L. Hagen and A. B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering", *IEEE Trans. on CAD* 11(9), Sept. 1992, pp. 1074-1085.

[13] K.M. Hall, "An r-dimensional Quadratic Placement Algorithm", *Manag. Sci.*, 17(1970), pp.219-229.

[14] B. W. Kernighan and S. Lin, "An Efficient Heuristic for Partitioning Graphs", *Bell Syst. Tech. J.* 49(2) (1970), pp.291-307.

[15] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley-Teubner, 1990.

[16] H. Nagamochi and T. Ibaraki, "Computing Edge-Connectivity in Multigraphs and Capacitated Graphs", *Siam J. of Disc. Math.* 5(1), Feb. 1992, pp. 54-66.

[17] D. Peleg and A. Schaffer, "Graph Spanners", *J. of Graph Theory* 13, 1989, pp. 99-116.

[18] S. Pissanetsky, *Sparse Matrix Technology*, Academic Press Inc., 1984.

[19] B. M. Riess, K. Doll, and F. M. Johannes, "Partitioning Very Large Circuits Using Analytical Placement Techniques", to appear in *Proc. ACM/IEEE Design Automation Conf.*, San Diego, 1994.

[20] L.A. Sanchis, "Multiple-way Network Partitioning", *IEEE Trans. on Computers*, 38, 1989, pp. 62-81.

[21] W. Sun and C. Sechen, "Efficient and Effective Placements for Very Large Circuits" *Proc. IEEE Intl. Conf. on Computer-Aided Design*, Santa Clara, Nov. 1993, pp. 170-177.

possible, we set $T = 1$ and $W = \frac{L+U}{2}$, creating a highly efficient methodology (e.g., a Sun Sparc 10 required 28.2 and 157.3 CPU seconds to generate the respective Primary2 and Biomed partitionings). Because DP-RP requires $O(n(U-L))$ memory, we were unable to construct partitionings for Industry2. We observed 22.5% and 31.5% Scaled Cost savings over EIG1 and SFC respectively.

			Algorithm		
Test Case	L	U	EIG1	SFC (d=5)	WINDOW
Primary1	133	200	77.72	84.72	76.09
Primary2	482	723	16.09	25.43	16.28
Test02	266	399	44.95	58.07	37.97
Test03	257	386	28.26	33.49	27.09
Test04	242	364	48.61	50.89	23.92
Test05	415	623	14.98	16.48	13.35
Test06	280	420	51.72	80.62	21.37
19ks	455	682	18.34	25.01	18.65
bm1	141	212	66.91	77.14	67.52
Biomed	1027	1540	16.00	N/A	2.23

Table 4: Scaled Costs for 5-way partitionings generated for EIG1, SFC, and WINDOW using DP-RP for Scaled Cost.

In conclusion, we have developed a general framework for constructing vertex orderings, and explored its applications to netlist clustering and partitioning. By setting an appropriate "attraction" function and window size, we obtained superior clusterings for a range of clustering objectives in the literature. We leave open the question of finding the improved attraction functions for meta-objectives that cannot be defined explicitly, such as two-phase FM enhancement.

Acknowledgements

Much thanks to Lars Hagen for supplying FM code, the RW-ST and BUI clusterings, and eigenvectors for Biomed and Industry2.

References

[1] C. J. Alpert and L. Hagen, *personal communication*, April 1994.

[2] C. J. Alpert and A. B. Kahng, "Multi-way Partitioning Via Spacefilling Curves and Dynamic Programming," to appear in *Proc. ACM/IEEE Design Automation Conf.*, San Diego, 1994 (extended paper released as UCLA tech. report #930016, 1993).

[3] C. J. Alpert and A. B. Kahng, "Geometric Embeddings for Faster and Better Multi-way Netlist Partitioning," *Proc. ACM/IEEE Design Automation Conf.* 1993, pp. 743-748.

Test Case	Algorithm					
	King	Cuthill-McKee	Max-Adjacency	EIG1	SFC	WINDOW
Primary1	209.6	226.7	176.1	244.1	204.7	173.1
Primary2	68.97	73.24	61.08	78.82	68.10	57.69
Test02	118.1	129.6	100.3	137.4	131.4	97.02
Test03	119.6	130.1	97.28	132.1	126.5	91.50
Test04	132.3	143.9	109.2	155.4	147.7	100.2
Test05	73.27	83.97	60.42	85.08	76.06	55.28
Test06	121.8	127.5	107.0	131.5	142.8	106.9
19ks	63.12	69.79	50.18	73.58	66.37	47.51
bm1	168.3	185.9	132.8	184.8	146.2	137.8
Biomed	28.26	30.41	22.69	35.26	N/A	22.06
Industry2	15.48	17.22	13.43	18.60	N/A	12.68

Table 2: Scaled Cost values of the clusterings DP-RP generated from the six given ordering constructions. Clustering sizes were generated for $L = 1$ and $U = 20$ using DP-RP with k same as in Table 1.

Test Case	Algorithm					
	King	Cuthill-McKee	Max-Adjacency	EIG1	SFC	WINDOW
Primary1	489.0	259.6	534.6	312.8	513.3	687.6
Primary2	1209	598.6	1355	600.2	1114	2257
Test02	709.8	282.9	991.4	468.8	618.6	1327
Test03	649.3	300.6	951.7	525.5	640.3	1247
Test04	716.8	326.8	992.3	382.6	625.3	1303
Test05	1069	490.8	1562	891.9	1324	2279
Test06	546.5	247.6	765.3	264.9	349.1	1274
19ks	1389	636.2	2042	806.2	1573	2556
bm1	512.8	278.0	546.4	355.5	513.8	692.5
Biomed	2083	1416	3592	5008	N/A	5036
Industry2	4038	1665	7072	10160	N/A	10338

Table 3: Absorption values of the clusterings DP-RP generated from the six given ordering constructions. Clustering sizes were generated for $L = 1$ and $U = 20$ using DP-RP with k same as in Table 1.

Our final set of experiments considered constructing a 5-way partitioning to optimize Scaled Cost such that the clusters are relatively equally sized. We chose $k = 5$ to make it difficult to apply a recursive multi-way partitioning algorithm[5]. The Scaled Cost values derived are reported in Table 4. The average cluster size is $\frac{n}{5}$ and we arbitrarily permitted $\frac{n}{25}$ deviation from the mean size for each cluster by setting $L = \frac{4n}{25}$ and $U = \frac{6n}{25}$. We ran DP-RP on EIG1, SFC, and WINDOW to derive the partitionings from the orderings. We only used one SFC ordering (derived from a five dimensional embedding) in order to make the comparisons more equitable. To make WINDOW as efficient as

[5]We note that the multi-way FM algorithm due to Sanchis [20] might perform well; however, Sanchis' original code cannot handle problem instances of size more than a few hundred.

Test Case	Algorithm	Scaled Cost	Absorption	DS	FM Cuts
Primary1	WINDOW	173.1	687.6	1.471	48
833	RW-ST	287.9	629.9	1.325	47
(191)	AGG	277.9	437.0	0.879	49
	MBC	254.0	309.3	1.258	48
Primary2	WINDOW	57.69	2257	1.539	186
3014	RW-ST	82.81	2013	1.566	165
(702)	AGG	89.73	1227	1.048	146
	MBC	82.44	736.4	1.238	187
Test02	WINDOW	97.02	1327	1.662	42
1663	RW-ST	150.7	1123	1.593	42
(445)	AGG	164.7	706.2	0.657	42
	MBC	137.4	407.0	1.231	42
Test03	WINDOW	91.50	1247	1.736	53
1607	RW-ST	156.2	1101	1.566	71
(327)	AGG	153.9	678.4	1.204	50
	MBC	140.7	379.5	1.185	59
Test04	WINDOW	100.2	1303	2.014	20
1515	RW-ST	151.8	1181	1.879	14
(317)	AGG	193.3	833.9	1.135	12
	MBC	160.3	415.5	1.297	20
Test05	WINDOW	55.28	2279	1.831	36
2595	RW-ST	88.52	2051	1.689	28
(424)	AGG	103.0	1527	1.262	32
	MBC	90.08	680.7	1.275	37
Test06	WINDOW	106.9	1274	1.516	73
1752	RW-ST	178.7	979.2	1.367	82
(476)	AGG	163.2	359.0	1.183	63
	MBC	142.4	315.3	1.331	83
19ks	WINDOW	47.51	2556	1.883	127
2844	RW-ST	81.08	2395	1.578	146
(737)	AGG	86.50	1485	1.022	124
	MBC	75.50	719.9	1.166	156
bm1	WINDOW	137.8	692.5	1.278	62
882	RW-ST	258.5	637.9	1.221	58
(216)	AGG	266.0	426.6	0.813	48
	MBC	340.5	199.1	1.189	54

Table 1: Comparison between WINDOW and four other clustering algorithms. The Primary1 and Primary2 FM results were generated with the SC area files. The numbers below each test case are n and k respectively.

clusterings with lowest Scaled Cost and highest absorption, though EIG1 was very competitive. The overall poor results for EIG1 and SFC do not mean these are not useful orderings, but these "global" orderings do not make the local decisions necessary for a good DP-RP generated clustering. They are still useful for multi-way partitioning, especially when there are no user-imposed sized constraints. In this case, it is not obvious what W and T values will optimize the WINDOW constructed ordering.

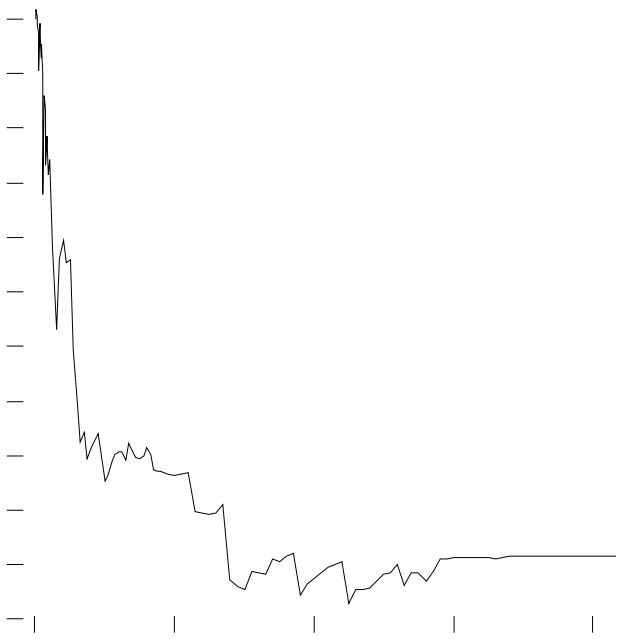

Figure 6: Absorption value shown as a function of window size W for Primary1. Clusterings were generated with $L = 1$, $U = 20$, $k = 191$, and $v = 15$. The attraction function was chosen to maximize Absorption.

constructed by each algorithm for each test case, again with $L = 1$, $U = 20$ and k as specified in Table 1. The respective Scaled Cost and Absorption values for the DP-RP generated clusterings are given in Tables 2 and 3. There were ten SFC orderings available for each test case (corresponding to embedding dimensions one through ten), and we report the best Scaled Cost and Absorption values derived over all of the orderings.

For the first nine listed test cases, we observed uniformly better results for both metrics, with the exception of Scaled Cost for bm1. In terms of Scaled Cost, we observed 4.0% improvement over WINDOW's next closest competitor, Max-Adjacency. However, Max-Adjacency is similar to WINDOW, but has a different attraction function and has $W = n$. Figure 6 showed that the range of values for different W is not very large for Absorption, and the range is even smaller for Scaled Cost. Discounting Max-Adjacency, we observed 18.3% average reduction in Scaled Cost over the best combined results of the other algorithms. In terms of Absorption, WINDOW observed 39.5% average increase over Max-Adjacency and 78.3% over the combined efforts of the other four ordering constructions. We also generated results for some larger test cases (Biomed: $n = 6417$, $k = 1283$, $L = 1$, $U = 20$; and Industry2: $n = 12142$, $k = 2428$, $L = 1$, $U = 20$) and WINDOW produced

For the WINDOW clusterings, we self-imposed cluster size constraints $L = 1$ and $U = 20$ in order to generate fast clusterings. We used $W = \lceil \frac{n}{k} \rceil$ and $T = U - W$ to generate the orderings. Although these may not necessarily be the optimum parameters, they are the most intuitive, as discussed in Section 3. Indeed, as Figure 6 illustrates, it is unclear what the exact relationship is among W, T and the clustering derived with these parameters. We computed a *pseudo-peripheral* vertex to be the first one chosen for the ordering.[4] For the Scaled Cost and Absorption objectives, we used the corresponding attraction functions of Section 2 and applied DP-RP to minimize the objective. Thus, for a given WINDOW row in Table 1, the Absorption clusterings and the Scaled Cost clusterings are not the same: when WINDOW constructs a clustering to optimize one objective, the clustering is usually inferior in terms of other objectives. For example, for Primary1, when minimizing Scaled Cost (173.1), the Absorption was 621.9 and when maximizing Absorption (687.6), the Scaled Cost was 234.8. Since it is unclear what the best attraction functions for DS and two-phase FM are, and since DP-RP cannot be applied to optimize these objectives, we measured DS and FM cuts in terms of the WINDOW clusterings that optimized Scaled Cost.

For Scaled Cost, Absorption and DS, WINDOW clusterings averaged 34.2%, 13.2% and 8.3% respective improvement over the next closest competitor. Moreover, the results were consistently better: the WINDOW-derived values were superior for every instance with the exception of Primary2 and the DS metric. Our FM results were competitive with MBC and RW-ST, though inferior to the 200 runs of AGG. We believe that these results can be improved upon when WINDOW is run with an attraction function more appropriate for two-phase FM, though it remains unclear what this function should look like. Our methodology was also extremely efficient; on a Sun Sparc 10, Primary2 required 9.7 CPU seconds to generate an ordering and DP-RP took an additional 106 seconds to generate the clustering. We also ran WINDOW on Biomed and Industry2 (see results given in the next subsection), and they required 36 and 63 seconds to generate their respective orderings, and 385 and 1322 seconds for DP-RP to construct the clusterings.

5.2 WINDOW Versus Other Vertex Orderings

We now compare the WINDOW orderings to five other vertex ordering constructions: King, Cuthill-McKee [18], Max-Adjacency [16], EIG1 [12] and SFC [2]. We ran DP-RP on the vertex ordering

[4]The *eccentricity* of a vertex v is the distance of the vertex u furthest from v. A *pseudo-peripheral vertex* v has the property that if the distance from u to v is the same as the eccentricity of v, then the eccentricity of u must be no greater than the eccentricity of v.

Cluster_Costs (Absorption)	
Input:	Linear ordering $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$
	$L, U \equiv$ Lower and upper cluster size bounds
Output:	$w(C_{[i,j]}) \equiv \sum_{(e \in E \mid e \cap C_i \neq \emptyset)} \frac{\lvert e \cap C_i \rvert - 1}{\lvert e \rvert - 1}$
	for every possible cluster $C_{[i,j]}$
Vars:	$p[e] \equiv$ the number of pins of e in the current cluster
	$S_j \equiv \{e \mid v_{\pi_j} \in e\}$
1. **for** $i = 1$ **to** n **do**	
2. $w(C_{[i,i]}) = 0$	
3. $p[e] = 0$ for all $e \in E$	
4. **for** $j = i$ **to** $\min\{i + U, n\}$ **do**	
5. **if** $j \neq i$ **then** $w(C_{[i,j]}) = w(C_{[i,j-1]})$	
6. **for** each $e \in S_j$ **do**	
7. $p[e] = p[e] + 1$	
8. **if** $(p[e] > 0)$ **then**	
$w(C_{[i,j]}) = w(C_{[i,j]}) + \frac{1}{\lvert e \rvert - 1}$	

Figure 5: Cluster_Costs algorithm for the Absorption metric.

5 Experimental Results

5.1 WINDOW Versus Previous Clustering Methods

Our first experiments compare the DP-RP clusterings derived from WINDOW vertex orderings with the MBC [4], RW-ST [11], and AGG [3] clustering methods. We evaluate the clusterings in terms of the four metrics discussed previously: DS, Scaled Cost, Absorption, and two-phase FM (min-cut bisection). The last, of course, is a "meta-objective". The results of the comparisons are given in Table 1. Below each test case, we list the number of modules n in the test case and the number of clusters k in parentheses. We use the same values for k as in the experiments of [11] [3] to facilitate comparisons.

The clusterings for MBC, RW-ST and AGG [3] were obtained from the authors [1]. There were ten AGG clusterings available for each test case, corresponding to geometric embeddings of dimension one through ten. We report the best AGG Scaled Cost and Absorption for the ten clusterings and the best min-cut from 200 FM runs (20 for each clustering). Due to the very high complexity of evaluating the DS metric, we only report the DS value for the AGG clustering with lowest Scaled Cost. The FM min-cuts for the other algorithms are the best observed from 20 runs and are quoted from [3] with the exception of the Test05 example, for which FM min-cuts were regenerated due to a faulty area file used in the original experiments.

the best 2-way partitioning solution over $\{v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_j}\}$. This will be a partitioning of the form $\{C_{[1,q]}, C_{[q+1,j]}\}$. DP-RP stores the best 2-way partitioning solutions for each j and then similarly derives the best 3-way partitionings (e.g., $\{C_{[1,i]}, C_{[i+1,q]}, C_{[q+1,j]}\}$, where q can range from 1 to j and i is determined from the best 2-way partitioning solution over $\{v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_q}\}$). The optimality of DP-RP follows from the monotonicity of the function f. For completeness, we reproduce the DP-RP algorithm [2] in Figure 4.

DP-RP Algorithm	
Input:	Linear Ordering $\{v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}\}$
	$L, U \equiv$ Lower and upper cluster size bounds
	$k \equiv$ Number of clusters
Output:	$P^k \equiv$ Restricted k-way partitioning solution
Vars:	$k' \equiv$ Index denoting current partitioning size
	$q+1 \equiv$ Beginning index of possible new cluster
1. $\forall i, j$ with $L \leq j - i + 1 \leq U$ compute $w(C_{[i,j]})$ using **Cluster_Costs**	
2. **for** $k' = 2$ **to** k **do**	
3. **for** $j = 1$ **to** n **do**	
4. Find the best k'-way partitioning for $\{v_{\pi_1}, v_{\pi_2} \ldots v_{\pi_j}\}$ composed of the best $k'-1$ partitioning for $\{v_{\pi_1}, v_{\pi_2} \ldots v_{\pi_q}\}$ and cluster $C_{[q+1,j]}$ for $j - U \leq q \leq j - L$	
5. **return** $P^k =$ the k-way partitioning derived for $\{v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}\}$	

Figure 4: The DP-RP algorithm of [2].

The efficiency of DP-RP requires an efficient implementation of the Cluster_Costs procedure. In [2], an $O(nU)$ implementation of Cluster_Costs was presented for the Scaled Cost metric. We now present the new result that the Absorption metric can also be optimized by DP-RP by presenting an $O(nU)$ implementation of Cluster_Costs in Figure 5. For a given $w(C_{[i,j]})$, Cluster_Costs initially assigns $w(C_{[i,j]}) = w(C_{[i,j-1]})$ thereby giving credit for the nets "absorbed" by modules $v_{\pi_i}, v_{\pi_{i+1}}, \ldots, v_{\pi_{j-1}}$. Steps 6-8 compute the number of nets absorbed when v_{π_j} is added to $C_{[i,j-1]}$. Steps 6-8 require constant time and are executed exactly pU times overall where p is the number of pins in the netlist. If the number of nets is approximately the number of modules and the average net size is a constant, then Cluster_Costs for Absorption has $O(nU)$ time complexity.

The WINDOW Algorithm	
Input:	Netlist $H(V, E)$ $W \equiv$ window size $T \equiv$ tail of the window Objective function *Attract*
Output:	Vertex ordering $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$
1. Choose a vertex $v_{i^*} \in V$ and set $v_{\pi_1} = v_{i^*}$ **for** each unordered $v_i \in V$, *do* compute *Attract*(i).	
2. **for** $index = 2$ **to** n **do**	
3. Choose unordered v_{i^*} such that *Attract*(i^*) is optimal	
4. Set $v_{\pi_{index}}$ to v_{i^*}	

5. **for** each unordered v_i update *Attract*(i) such that
if $index - W + 1 \leq j \leq index$, **then**
vertex v_{π_j} has full attraction on v_i
if $index - W - T + 1 \leq j \leq index - W$, **then**
vertex v_{π_j} has $\frac{index-W-i}{T}$% attraction on v_i	

6. **return** $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$	

Figure 3: The WINDOW Algorithm

the updating of a vertex takes time proportional to T, and the complexity becomes $O(nT|N|)$.

4 The DP-RP Algorithm for Clustering Orderings

The work of [2] presented the "DP-RP" algorithm for constructing a multi-way partitioning. The algorithm solves a *restricted partitioning* formulation that accepts a vertex ordering as input and returns a k-way partitioning such that each cluster is a contiguous subset of the ordering. This idea occurs elsewhere in the literature, e.g., the EIG1 algorithm [12] applied this restricted formulation to 2-way partitioning and used the best of the $n-1$ possible splits of the eigenvector ordering. Although the number of possible restricted k-way partitioning solutions increases exponentially with k, DP-RP uses dynamic programming to find the optimal set of $k-1$ splits that induce the k-way partitioning. DP-RP constructs its solutions in $O(kn(U-L))$ time for any clustering objective that is expressible as a *monotone* function (e.g., sum, min, and max) of an intercluster weight metric. We have seen that both Absorption and Scaled Cost are both expressible as a monotone function in w.

In the k-way restricted partitioning formulation, since each cluster must be a contiguous subset of the vertex ordering, there are at most nU clusters that can be in P^k. Every such cluster can be uniquely denoted by $C_{[i,j]} = \{v_{\pi_i}, v_{\pi_{i+1}}, \ldots, v_{\pi_j}\}$ where $j - i < U$. DP-RP first computes $w(C_{[i,j]})$ for each possible cluster via a Cluster_Costs procedure. Observe that cluster $C_{[1,j]}$ is the unique 1-way partitioning solution over $\{v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_j}\}$. By considering all possible q, with $1 \leq q \leq j$, we can find

"bottom" of the window. Figure 2 illustrates the attraction percentages exerted by the first 100 ordered vertices for different sized windows and tails. The choice for W and T can significantly influence the degree of attraction exerted on an unordered vertex. Figure 3 formally integrates the concepts of a window and tail into the WINDOW vertex ordering construction.

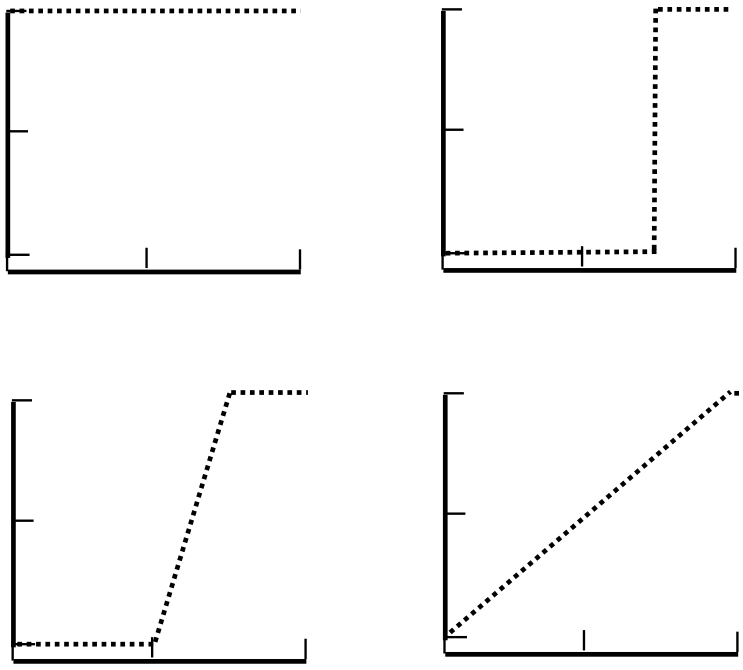

Figure 2: The percent attraction exerted by ordered vertices on unordered vertices for various W and T after the first 100 vertices have been ordered. (a) $W = n$, $T = 0$ corresponds to the unbounded window discussed Section 3; (b) $W = 25$, $T = 1$ coresponds to a window with no tail; (c) $W = 25$, $T = 25$ corresponds a window with a tail; and (d) $W = 1$, $T = n$ corresponds to a "history window" where each subsequently ordered vertex exerts more attraction than previously ordered vertices.

If $T = 1$ (i.e., a window without a tail), then WINDOW can be implemented in linear time for many attraction functions, assuming W is a constant. Let N be the set of unordered neighboring vertices to the window. Since netlist modules have bounded degree, $|N|$ is a constant proportional to W. During WINDOW's execution, every best vertex candidate will be in N; hence, the Step 5 updating can be done by adding unordered neighbors of the chosen v_{i*} to N, and then updating *Attract*(i) for each $v_i \in N$. To update *Attract*(i) for a given v_i, we add the attraction exerted by v_{i*} on v_i to *Attract*(i) and subtract from *Attract*(i) the attraction exerted on v_i by $v_{\pi_{index-W}}$, the vertex just outside the new window. Hence, WINDOW runs in $O(n|N|)$ time when $T = 1$. If $T \neq 1$, then

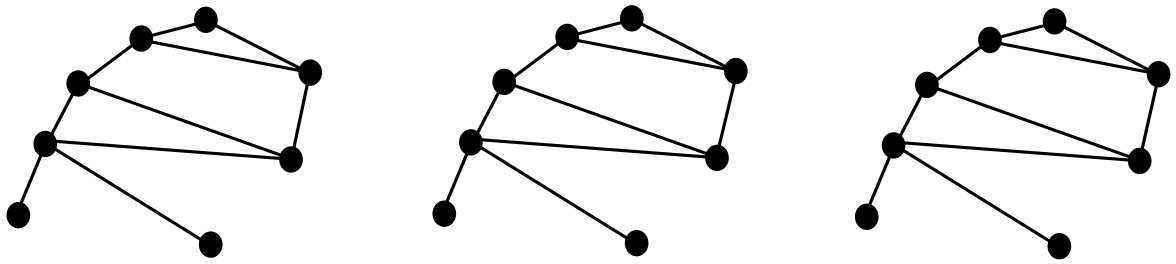

Figure 1: Snapshots for feasible orderings for (a) DFS, (b) BFS, and (c) Max-Adjacency. An ordered vertex is labelled by its corresponding index, and an unordered vertex is labelled with its attraction A. For (a) and (c) the best vertex has maximum A; for (b) the best vertex has minimum A.

clustering. Our intuition for a good ordering requires that a small contiguous segment of the ordering should form a good cluster. It follows that the vertices ordered early will not share this contiguous cluster with vertices ordered later, especially if the segment is small. Hence, when choosing v_{i*}, only the ordered vertices that might share a cluster with v_{i*} should exert attraction on it. Instead of every vertex in S having influence on the choice for v_{i*}, we propose that only the most recently ordered vertices should have influence. To this end, we propose the following WINDOW algorithm.

3 The WINDOW Vertex Ordering Construction

We define the current *window* of size W to be the set of the W most recently ordered vertices and we will often refer to the window as the "current cluster". Only vertices in the window can exert full attraction on unordered vertices. For constructing a k-way clustering over n modules, the user could logically set W to be $\frac{n}{k}$, the average cluster size. When the ordering is later split into its k clusters (by DP-RP), a window of this size enables every window vertex to possibly share a cluster with the next best vertex v_{i*}. However, this framework alone cannot adequately reflect the cluster size bounds. For example if $W = \frac{n}{k} = 5$ and $L = 1$, then this framework will construct the same ordering for $U = 10$ as for $U = 1000$. In the latter case, as many as 994 ordered vertices may share a cluster with v_{i*} yet will have no influence in the choice of this vertex.

To accommodate cluster size bounds, we let T be the size of the *tail* of the window, where vertices in the tail have some influence on the choice for v_{i*} but not to the same extent as the window vertices. The attraction exerted by a vertex in the tail is proportional to the distance the vertex is from the

- **The Absorption Metric.** If we regard S as a cluster, we can let the best vertex v_{i*} be such that $S \cup \{v_{i*}\}$ has maximum Absorption:

$$Attract(i) = \sum_{\{e \in Nets(i) \mid e \cap S \neq \emptyset\}} \frac{1}{|e| - 1}$$

  For each e incident to v_i and S, an absorption of $\frac{1}{|e|-1}$ is gained by adding v_i to S. Hence, the v_{i*} that maximizes $Attract(i^*)$ yields the greatest increase in the absorption of S.

- **The Scaled Cost Metric.** The Scaled Cost of a cluster is the outdegree of the cluster divided by its size, yet an attraction function chosen to minimize cluster outdegree will generally not yield a good ordering. For example, if 8 pins of a 10-pin net are already ordered, then the other 2 pins should have strong attraction to the cluster to prevent the net from being cut; however, under a strict perimeter attraction function, this net will exert no attraction on either pin's module until the one of the two modules becomes ordered. Hence, we used the following alternative attraction function for Scaled Cost which worked well in practice:

$$Attract(i) = \sum_{e \in Adj(i)} \frac{|S \cap e|}{|e| - 1}$$

  For the Absorption attraction function, a net e incident to v_i and S exerts the same attraction on v_i regardless of how many pins of e are in S. Under this Scaled Cost attraction function, a net $e \in Nets(i)$ exerts attraction on v_i proportional to the number of its pins in S. Thus, as more vertices of a given net become ordered, the unordered vertices incident to that net feel stronger attraction to the ordering.

For any attraction function, if the updating of $Attract(i)$ (Step 3) is implemented naively, then our framework may take up to $O(n^2)$ time to construct an ordering. However, for many attraction functions, $Attract(i)$ can never decrease (increase), for additional vertices added to S can only increase (decrease) the attraction exerted on unordered vertices. Our present implementation uses a Fibonacci heap to store each $v_i \in V - S$ with a corresponding key of $Attract(i)$, During each iteration, the vertex with maximum (minimum) key is extracted from the heap and the keys for the other vertices are updated via an increase-key (decrease-key) operation. With this implementation, our framework requires just $O(n \log n)$ amortized time.

The attraction functions for the Max-Adjacency ordering and for the Scaled Cost and Absorption metrics regard S as the "current cluster" though this will not be the case in our subsequent k-way

2. **Best Vertex:** If $V - S \neq \emptyset$, pick the vertex $v_{i^*} \in V - S$ such that $Attract(i^*)$ is optimal, otherwise exit.

3. **Update:** Increment $index$ and then set $\pi(index) = i^*$. Update $Attract(i)$ for each $v_i \in V - S$ and go to Step 2.

The ordering derived from this greedy approach will directly depend on the choice for the first vertex in the ordering and on the attraction function. We now give some examples of attraction functions that illustrate that virtually any traditional vertex ordering construction can be captured by our framework. Let $Nets(i) = \{e \in E \mid v_i \in e\}$ be the set of nets incident on v_i and let $Adj(i) = \{v_j \in e \cap S \mid e \in Nets(i)\}$ be the set of ordered neighbor vertices of v_i.

- **DFS Ordering.** We define the attraction for v_i as:

$$Attract(i) = \max\{j \mid v_{\pi_j} \in Adj(i)\}$$

  If all of v_i's neighbors are unordered then we set $Attract(i) = 0$. In other words, $Attract(i)$ is the index of the neighbor of v_i most recently added to the ordering. The "best" vertex is the v_{i^*} such that $Attract(i^*)$ is maximum, i.e., v_{i^*} will be adjacent to the most recently ordered vertex $v_{\pi_{index}}$ unless $v_{\pi_{index}}$ has no unordered neighbors. In this case, we backtrack to $v_{\pi_{index-1}}, v_{\pi_{index-2}}, \ldots$ until we find a $v_{\pi_{index-q}}$ that has unordered neighbors, one of which will be $v_{\pi_{i^*}}$. Figure 1(a) shows an example of the $Attract$ values during the construction of a DFS ordering, given that five vertices have already been ordered.

- **BFS Ordering.** We define the attraction for v_i as:

$$Attract(i) = \min\{j \mid v_{\pi_j} \in Adj(i)\}$$

  If v_i has no ordered neighbors then $Attract(i) = \infty$ and the best vertex is the v_{i^*} such that $Attract(i^*)$ is minimum. Figure 1(b) shows the attraction values for a sample BFS ordering.

- **Max-Adjacency Ordering.** For general graphs, the authors of [16] defined the attraction for v_i to be the number of edges $(v_i, v_j) \in E$ such that v_j is ordered. The best vertex v_{i^*} has the most edges incident to S, i.e., is maximally adjacent to S. For hypergraphs we extend the definition of the attraction of v_i to be the number of hyperedges incident to both v_i and S:

$$Attract(i) = |\{e \in Nets(i) \mid e \cap S \neq \emptyset\}|$$

  Figure 1 (c) shows an example of this attraction function.

Finally, there are several vertex orderings that are inferred from heuristic global minimization of wirelength functions in one-dimensional module placements. Hall [13] showed that the second eigenvector of the netlist discrete Laplacian yields the natural module ordering for minimum squared wirelength; we call this the EIG1 ordering, following [12], who used this eigenvector for ratio-cut partitioning. Recently, Riess et al. [19] have used an analytical conjugate gradient method to construct module orderings according to the linear wirelength objective. Their PARABOLI orderings improve ratio cut partitionings by 50% over EIG1 orderings. Also recently, Alpert and Kahng [2] have induced (one-dimensional) module orderings via spacefilling curves (SFCs) over multi-dimensional spectral netlist embeddings.

The remainder of the paper is organized as follows. Section 2 describes our new framework for vertex orderings which captures existing methods using the concept of an "attraction" function. In an iterative graph traversal, the next vertex in the ordering is selected based on the amount of attraction between unordered vertices and the set of ordered vertices. Given an upper bound on cluster sizes, vertices at one end of the ordering cannot share a cluster as vertices at the other end; we implicitly assume that the individual clusters will be contiguous subsets of the vertex ordering. Thus, Section 3 proposes the WINDOW algorithm, in which only a window of the W most recently ordered vertices exert attraction on the unordered vertices. To split a given vertex ordering into a k-way clustering, we apply the DP-RP dynamic programming approach of Alpert and Kahng [2]. DP-RP finds the optimal set of splits for many standard clustering objectives, and can handle user-specified cluster size bounds. Section 4 reviews the DP-RP algorithm and shows how it can be applied to the Absorption objective. Finally, we present several sets of experimental results in Section 5.

2 The Attraction Function

We propose the following graph traversal framework for constructing a vertex ordering. We say a vertex v_j has been assigned to the ordering, i.e., is *ordered*, if $\pi(index) = j$ for some $index$ and otherwise we say v_j is *unordered*. We will generally use v_j for an ordered vertex, v_i for an unordered vertex, and v_{i*} for the "best" unordered vertex. Let $S = \{v_j \in V \mid v_j \text{ is ordered } \}$ and, for each unordered v_i, let $Attract(i)$ be the *attraction* from v_i to S

1. **Initialize:** Choose a vertex v_{i*} and set $\pi(1) = i^*$. Set $index$, the current size of S, to 1. For each $v_i \in V - S$, compute $Attract(i)$.

1.2 Vertex Orderings

We represent a *vertex ordering* $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$ of vertices $V = \{v_1, v_2, \ldots, v_n\}$, by the bijection $\pi : [1 \ldots n] \rightarrow [1 \ldots n]$. Vertex v_i is the j^{th} vertex in the ordering if $\pi(j) = i$ (so $v_i = v_{\pi_j}$), i.e., v_{π_1} is the first vertex in the ordering, v_{π_2} is the second vertex, etc.

The Vertex Ordering Problem: Given $H(V, E)$, construct a vertex ordering $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$ to optimize some objective.

As with the k-way clustering formulation, the objective can range from well-defined to nebulous. However, intuition says that the objective should allow a good ordering to preserve "structure" in the netlist: strongly connected modules should be near each other in the ordering, and a contiguous subset of the ordering should likely form a "good cluster" according to the above metrics. Vertex ordering methods have arisen in many applications; the following provides a brief sampling.

For sparse matrix computations, rows of a symmetric n by n matrix are reordered to minimize the *bandwidth* or *profile* of the matrix, enabling efficient storage schemes (see [18] for a survey). Row reordering is also useful to reduce the *fill-in*, i.e., nonzeros added into the matrix when Gaussian elimination is performed. Since the non-zeros in a given matrix can be viewed as the adjacency matrix of a graph with n vertices, the row reordering problem corresponds directly to vertex ordering in a graph:

- The Cuthill-McKee algorithm for minimizing bandwidth is a BFS variant in which ties are broken in favor of the vertex with smallest degree.
- The King algorithm is a *min-perimeter* approach: it iteratively selects the vertex that minimizes the number of unordered vertices that are adjacent to ordered vertices. In other words, if we consider the set of ordered vertices as a single cluster, the King algorithm iteratively adds the vertex that minimizes the "perimeter" of the resulting cluster.

For constructing sparse graph spanners, Peleg and Schaffer have applied a *max-perimeter* approach. For fast computations of edge connectivity in unweighted graphs, Nagamochi and Ibaraki [16] have proposed a *max-adjacency* ordering: their algorithm iteratively adds the vertex which has the most edges to vertices that are already in the ordering[3].

[3]In VLSI CAD, the min-perimeter and max-adjacency orderings are at the heart of "direct" clustering methods used in early placement and partitioning approaches [15]. Similar orderings have been applied to the one-dimensional gate assignment problem (e.g., [9]).

- Cong et al. [7] proposed the **DS** objective:

$$maximize \quad f(P^k) = \frac{1}{n} \sum_{i=1}^{k} w(C_i)$$

where

$$w(C_i) = |C_i| \cdot \frac{degree(C_i)}{separation(C_i)}$$

Here, $degree(C_i)$ is the average number of nets incident to each module of the cluster and having at least two pins in the cluster; $separation(C_i)$ is the average length of a shortest path between two modules in C_i, with separation $= \infty$ if two modules in the cluster are disconnected. The DS metric attempts to capture intuitive aspects of a good clustering, but takes $O(n^3)$ time to evaluate, making it useful only for comparing the quality of two clustering solutions – it cannot be directly optimized efficiently.

- Sun and Sechen [21] proposed a new clustering objective that we call **Absorption**, since it counts the number of nets which are "absorbed" by the clusters:

$$maximize \quad f(P^k) = \sum_{i=1}^{k} w(C_i)$$

where the absorption $w(C_i)$ of a cluster is

$$w(C_i) = \sum_{\{e \in E \mid e \cap C_i \neq \emptyset\}} \frac{|e \cap C_i| - 1}{|e| - 1}$$

If a net e is incident to cluster C_i, then it adds absorption $= (p-1) \cdot \frac{1}{|e|-1}$ to the cluster, where p is the number of pins of e in the cluster. For example, if all pins of e are in C_i, then e contributes absorption $= 1$ to C_i. We can view this objective another way: if the i^{th} net e_i has its pins distributed among c_i clusters, then $f(P^k) = \sum_{i=1}^{m} \frac{|e_i|-c_i}{|e_i|-1}$.

- Finally, **Scaled Cost** was proposed by Chan et al. [5] as the k-way generalization of the ratio cut objective:

$$minimize \quad f(P^k) = \frac{1}{n(k-1)} \sum_{i=1}^{k} w(C_i)$$

where the cost of a cluster is

$$w(C_i) = \frac{|\{e \mid \exists u, v \in e, u \in C_i, v \notin C_i\}|}{|C_i|}$$

In other words, $w(C_i)$ is the "outdegree" of a cluster, divided by its size.

L and U, construct $P^k = \{C_1, C_2, \ldots, C_k\}$ with $L \leq |C_i| \leq U$, $1 \leq i \leq k$, that optimizes a given objective function $f(P^k)$.

We loosely refer to P^k as a *clustering* when k is large, e.g., $k = \Theta(n)$; P^k may be referred to as a *partitioning* when k is small, e.g., $k \leq 10$. Where possible, we will let $w(C_i)$ denote the *cost* or weight of having C_i in the clustering, and express f in terms of w.

1.1 Clustering Methods and Metrics

Bui et al. [4] proposed the linear-time Matching Based Compaction (MBC) algorithm, which finds a random maximal matching in the netlist and "compacts" the matched pairs of modules into $\frac{n}{2}$ clusters. The random matching can be repeated if desired. Hagen and Kahng [11] suggested using a random-walk (RW-ST) to find dense areas of the netlist, with "cycles" in the random walk forming the clusters. The suggested implementation requires $O(n^3)$ time. In another direction, Alpert and Kahng [3] and Chan et al. [5] apply geometric or hybrid geometric-topological clustering algorithms to multi-dimensional spectral embeddings of the netlist. Generating the eigenvectors that yield the embedding can be expensive (see also [2]), and such approaches can be complicated. Cong and Smith [6] present a bottom-up clustering method that recursively finds and collapses densest subgraphs of small size in the netlist. Although the method is amenable to parallel processing, any implementation can be expensive due to the complexity of clique-finding. Finally, the recent work of Sun and Sechen [21] applied clustering and the two-phase approach to simulated annealing placement. Their clustering algorithm starts with modules assigned to k "bins" or clusters and then uses 100 runs of "linear-time simulated annealing" to reassign the modules among bins. The method also integrates user-specified upper and lower cluster size bounds. All of these methods center on "meta-objectives" (i.e., the utility of the clustering within two-phase FM bisection, or within two-phase annealing placement). Moreover, they share a common "static" nature: it is not clear how to adapt them to construct a different clustering if some other objective is desired.

A number of works have proposed explicit objectives for netlist clustering. Of particular note[2]:

[2]In addition to the three objectives that we discuss, Garbers et al. [10] have proposed a k-l connectivity criterion. Two vertices are k-l-connected if there are k edge-disjoint paths of length $\leq l$ between them. While there is some flexibility in that k and l are specified by the user, evaluating the objective is infeasible for $l > 2$ (and is known to be NP-hard for $l > 4$).

ordering can wander, rather than remain in a dense region; and (ii) a BFS ordering will visit all neighbors of a given vertex but then can jump to an entirely different region of the topology. Thus, we first develop a new framework for traversing a graph to induce a vertex ordering.

Within our framework, vertices are added into the ordering based on their *attraction* to the previous history of the ordering. This paradigm is extremely flexible – given particular attraction functions, it can capture depth-first, breadth-first, max-adjacency [16], min-perimeter [17], and many other well-studied orderings – and generally affords a linear-time computation of the vertex ordering. The user can also set different variables to construct an ordering best suited for a particular application; we believe our methods have applications to problem classes such as the ordering of one-dimensional logic arrays, netlist partitioning, module placement, and even sparse matrix computations. However, this work centers on applications to netlist clustering, for the the following reasons.

- By reducing the problem size, clustering significantly improves Fiduccia-Mattheyses (FM) bisection via the "two-phase" methodology [4] where FM is first run on the clustered netlist, then run a second time on the flattened netlist. Sun and Sechen [21] applied two-phase clustering to simulated annealing placement, also with good results.[1]
- There is a lack of fast, high-quality clustering constructions in the literature. Most existing approaches are fairly expensive, and cannot be adapted to alternate objectives.
- Most critically, it is unclear what the appropriate clustering objective might be. For such "meta-objectives" as the clustering's effectiveness within two-phase FM, there may be many good solutions, and hence many plausible objectives should be explored. A fast and flexible methodology is necessary.

We formalize the k-way clustering problem as follows. We assume we are given a netlist hypergraph $H(V, E)$ which consists of a set of modules (vertices) $V = \{v_1, v_2, \ldots, v_n\}$ and a set of nets (hyperedges) $E = \{e_1, e_2, \ldots, e_m\}$. A net e_j is a subset of V with $|e_j| \geq 2$, a *cluster* C_i is a nonempty subset of V, and a *k-way clustering* P^k is a set of k clusters such that every $v_i \in V$ belongs to exactly one cluster in P^k

The k-Way Clustering Problem: Given $H(V, E)$, a value $2 \leq k \leq n$, and cluster size bounds

[1] Such efforts confirm the intuition of Bui et al. [4] and Lengauer [15] that good local minima in the flat instance are more easily reached from good local minima in the clustered instance.

A General Framework for Vertex Orderings, With Applications to Netlist Clustering*

C. J. Alpert and A. B. Kahng

UCLA Computer Science Department, Los Angeles, CA 90024-1596

Abstract

We present a general framework for the construction of *vertex orderings*, encompassing virtually all reasonable graph traversals. We study these vertex orderings for netlist clustering, which is a basic operation for both netlist partitioning and module placement. Many previous clustering algorithms, e.g., [4] [11] [6] [3], are static in that they generate the same (or similar) solutions for a given netlist, regardless of the cluster size constraints and the objective function. By contrast, our framework is simple and general: we iteratively visit modules and assign them into the ordering such that the module with the highest *attraction* to the already-ordered modules is added next into the ordering. The variant choices for the attraction function allows our framework to subsume most graph traversals and address a range of clustering objectives. To address user-specified cluster size bounds, we propose a modification of our framework: the WINDOW algorithm. WINDOW constructs a vertex ordering based on the attraction of unordered vertices to only the W most recently ordered modules, i.e., modules in our current *window*. Since the most recently ordered module is the best one to add to the current "window" cluster, we set W to be the average cluster size. The DP-RP method of [2] can then be applied to optimally split the vertex ordering into a k-way clustering. Our WINDOW + DP-RP derived clusterings are superior to a number of clustering methods in the literature in terms of three distinct objectives [5] [21] [7]. The WINDOW orderings, by themselves, are also superior to previous graph ordering constructions [12] [16] [18] [2]. Results for both clustering and multi-way partitioning are quite encouraging.

1 Introduction: Clusterings and Vertex Orderings

We seek to construct vertex orderings that *capture the clustering structure* of a netlist hypergraph, i.e., if we isolate any contiguous subset of the ordering, its vertices will form a "good" (e.g., connected, dense, etc.) cluster. For this purpose, the vertex orderings induced by fundamental graph traversals such as depth-first search (DFS) or breadth-first search (BFS) are generally insufficient: (i) a DFS

*Partial support for this work was provided by a Department of Defense Graduate Fellowship, and by NSF MIP-9257982 and MIP-9223740.