Seed-Growth Heuristics for Graph Bisection

Harvard CS TR-10-99

Wheeler Ruml
Division of Engineering and Applied Sciences
Harvard University
ruml@eecs.harvard.edu

Joe Marks
MERL–A Mitsubishi Electric Research Laboratory
marks@merl.com

Stuart M. Shieber
Division of Engineering and Applied Sciences
Harvard University
shieber@eecs.harvard.edu

J. Thomas Ngo
Interval Research Corporation
ngo@interval.com

March 31, 1998*

*This report is an expanded version of a paper presented at *Workshop on Algorithms and Experiments, 1998* in Trento, Italy. Due to delays involving consideration of this report for journal publication, this technical report was not published until November, 1999 or made available electronically until May, 2000.

Abstract

We investigate a family of algorithms for graph bisection that are based on a simple local connectivity heuristic, which we call *seed-growth*. We show how the heuristic can be combined with stochastic search procedures and a postprocess application of the Kernighan-Lin algorithm. In a series of time-equated comparisons against large-sample runs of pure Kernighan-Lin, the new algorithms find bisections of the same or superior quality. Their performance is particularly good on structured graphs representing important industrial applications. An appendix provides further favorable comparisons to other published results. Our experimental methodology and extensive empirical results provide a solid foundation for further empirical investigation of graph-bisection algorithms.

Keywords: Graph bisection, heuristic algorithms, computer-aided design, graph partitioning.

1 Introduction

Given a graph $G = (V, E)$ with an even number of vertices, the graph bisection problem is to divide V into two equal-size subsets X and Y such that the number of edges connecting vertices in X to vertices in Y (the size of the *cut set*, notated $\mathrm{cut}(X, Y)$) is minimized. This problem is NP-complete [12]. Although we will not address generalizations of the problem in this paper, popular ones include dividing V into more than two subsets, relaxing the strict equality constraint on the sizes of the subsets, and associating weights with the vertices and edges. Graph bisection has considerable practical significance, especially in the areas of VLSI design and operations research.

The benchmark algorithm for graph bisection is due to Kernighan and Lin [19]. (Data structures allowing an efficient implementation of this heuristic technique were described by Fiduccia and Mattheyses [9], so the algorithm is sometimes referred to as the Kernighan-Lin-Fiduccia-Mattheyses algorithm.) The Kernighan-Lin (KL) algorithm improves an initial random bisection by making a sequence of locally optimal vertex swaps between the partitions X and Y. The vertex-swap operation is also the primitive perturbation operator used in applications of simulated annealing to graph bisection [20, 21]. In spite of the folk wisdom that the simulated-annealing search strategy is capable of avoiding the local minima that often plague greedy heuristics like the KL algorithm, Johnson et al. [18] found that the relative performance of the two algorithms depends on the nature of the

graphs being bisected. Although their conclusions were quite detailed, a rough summary is that simulated annealing has an advantage on relatively uniform graphs, but KL is better for graphs with structure.

Recently, more aggressive attempts have been made to exploit the structure that is often found in graphs of practical significance. The common theme of these attempts is clustering: by grouping together vertices in tightly connected subgraphs, clusters of vertices can be treated as individual supernodes during the application of standard heuristics like KL or simulated annealing. The various incarnations of the clustering idea appear to show a marked superiority over the original KL algorithm [4, 6, 8, 11, 13, 16, 22, 26, 27], though the degree of superiority is unclear because the reported empirical results tend to sell the KL algorithm short, as we will argue below.

The algorithms we present in this paper can be considered a synthesis of ideas from previous work. They are based on a simple constructive heuristic that, although it has shown some promise in previous work [25, 23, 2], has never been investigated thoroughly; in particular, it has not been compared fairly to KL. We investigate the use of a parallel hill-climbing search strategy to improve performance when given extra computation time, and the use of the KL algorithm for final refinement of the computed bisections. When compared fairly with the KL algorithm (i.e., giving each algorithm equal time and ensuring that a large sample of KL runs is considered), the new algorithms exhibit significant superiority on a variety of test graphs. Our experimental methodology and results may also serve as a basis for further investigation of related graph-bisection techniques.

In the following sections we describe our algorithms, present an empirical analysis of their behavior, and conclude with a discussion of future work. An appendix provides further comparisons with other algorithms.

2 Algorithm Descriptions

2.1 The Seed-Growth Heuristic

Our algorithms are based on a simple *seed-growth* heuristic, which uses only the local connectivity of the graph to guide its construction of a bisection. We start with two disjoint, equal-size subsets of the vertex set to seed the two partitions, and add the remaining vertices one at a time into alternate partitions, at each step choosing the vertex to be added in a greedy manner. When adding to partition X we choose a vertex a that minimizes $\mathrm{cut}(\{a\}, Y) - \mathrm{cut}(\{a\}, X)$, the *relative connectedness* of a to Y. Intuitively, we minimize the number of edges added to the cut set separating X and

Y while maximizing the number of edges barred from future addition to the cut set. Thus the exploitation of structure is implicit in this heuristic, as compared to heuristics in which explicit clusters are computed and manipulated [4, 6, 8, 11, 13, 16, 22, 26, 27].

More formally, the algorithm can be given by the following pseudocode. To simplify notation, we assume $|V|$ is even, although in practice we tolerate solutions with an imbalance of 1. The constant *size* specifies the cardinality of the seed sets.

Input: An undirected graph $G = (V, E)$.

Output: A partition of V into subsets X and Y of size $\frac{|V|}{2}$.

Procedure:

1. Let the *seed sets* s_x and s_y be randomly chosen disjoint subsets of V such that $|s_x| = |s_y| = size < \frac{|V|}{2}$.
2. $X \leftarrow s_x; Y \leftarrow s_y$.
3. Repeat substeps (a) and (b) until all the vertices in V have been assigned to X or Y:
   (a) Find an unassigned vertex $a \in V$ such that $\text{cut}(\{a\}, Y) - \text{cut}(\{a\}, X)$ is minimal.
   $X \leftarrow X \cup \{a\}$.
   (b) Find an unassigned vertex $b \in V$ such that $\text{cut}(\{b\}, X) - \text{cut}(\{b\}, Y)$ is minimal.
   $Y \leftarrow Y \cup \{b\}$.

This constructive heuristic can be implemented to run in $O(|V| + |E|)$ time using the bucket data structure of Fiduccia and Mattheyses [9]. Since it makes only a single pass through the vertices, it is roughly five times faster than an efficient implementation of the KL algorithm on standard test graphs. On average, a single execution of this heuristic will be worse than a single application of the KL algorithm, which may be why seed-growth techniques have been largely ignored until recently. But the heuristic can be rendered effective by running it many times as part of a general search procedure. We will examine two search techniques in this paper, random generate-and-test and parallel hill climbing, although others (e.g., simulated annealing and genetic algorithms) might also be used effectively.

2.2 Previous Work

The seed-growth heuristic can be viewed as a simplified descendent of the 'direct' hypergraph partitioning methods described by Donath [8], which date back to the late 60s. It has been independently developed at least twice and several algorithms incorporating it have been proposed, but to our knowledge, no thorough empirical analysis of it has been done before.

Laguna, Feo, and Elrod [23] presented an algorithm they call GRASP (for 'greedy randomized adaptive search procedure') which performs seed growth using a *size* of 1 and a slightly elaborated selection criterion: when there are very few vertices that have the minimum relative connectedness, a larger candidate set is created that also contains a few vertices with the next highest score. To improve solution quality, a mild-ascent hill-climbing procedure is used to refine the resulting bisections. To make use of additional computation time, the heuristic and refinement combination is run multiple times using different random seed vertices. Unfortunately, Laguna, Feo, and Elrod do not use the efficient data structures of Fiduccia and Mattheyses, and their empirical comparison to KL falls prey to the concerns we will discuss in the next section.

Dell'Amico and Maffioli [7] replaced the hill-climbing refinement of GRASP with two different variants of tabu search. In a time-equated comparison to Laguna, Feo, and Elrod, their EnTaS algorithm (for 'enhanced tabu search') resulted in an improvement on all random graphs tested except sparse structured ones, although the degree of improvement is hard to assess because the variance in performance of each algorithm on each instance is not reported. KL is not considered in their comparison.

Battiti and Bertossi [2] simplified GRASP to the bare seed-growth heuristic, which they call Diff-Greedy. Their algorithm does not use a refinement procedure on the seed-growth solutions or GRASP's expanded candidate list. Their implementation uses the efficient bucket data structure, but instead of comparing against KL, they present suggestive although incomparable results from EnTaS. We will examine the performance of Battiti and Bertossi's Diff-Greedy below. Since it can be viewed as combining the seed-growth (SG) heuristic with a simple random generate-and-test (RGT) search procedure, we will refer to it as RGT/SG. We will also investigate a novel variant in which we use the KL algorithm as a postprocess to refine the resulting bisections. That algorithm will be notated as RGT/SG+KL.

Independently, Marks et al. [25, 24] presented a version of seed-growth where $size = \lfloor \underline{0.01} |V| \rfloor$. For refinement, they use the KL algorithm. Instead of using multiple independent restarts with random seed sets, they

use a population-based hill-climbing method to incrementally modify seed sets resulting in good bisections. Their algorithm can be sketched as follows. Underlined quantities are parameters of the heuristic that can be varied; we present the values they recommend.

Input: An undirected graph $G = (V, E)$.

Output: A partition of V into subsets X and Y of size $\frac{|V|}{2}$.

Procedure:

1. Randomly choose a set P of $\underline{100}$ pairs (s_x, s_y) of seed sets using Step 1 of the seed-growth heuristic.
2. Compute the corresponding bisection (X, Y) for each seed-set pair $(s_x, s_y) \in P$ using Steps 2 and 3 of the seed-growth heuristic.
3. For each bisection (X, Y) that scores in the top $\underline{20\%}$, use the KL procedure to separately compute a refined bisection (X^r, Y^r), leaving the original unchanged. Record the best refined bisection found as B.
4. Repeat substeps (a) through (e) until the allotted computation time has expired:
   (a) Randomly pick a seed-set pair $(s_x, s_y) \in P$ according to a distribution which makes the best seed set $\underline{4}$ times as likely to be chosen as the worst, with uniform increments in between.
   (b) Randomly select a vertex in one of s_x or s_y and replace it with another randomly chosen seed vertex from $V - s_x \cup s_y$; call the resulting seed-set pair (s'_x, s'_y).
   (c) Compute the corresponding bisection (X', Y') using Steps 2 and 3 of the seed-growth heuristic.
   (d) Add (s'_x, s'_y) to P. If its bisection scores in the top $\underline{20\%}$, use the KL procedure to separately compute a refined bisection, and update B if this refined bisection is better.
   (e) Remove the worst seed set from P.
5. Return B.

Because this algorithm combines parallel hill climbing (PHC), the growth heuristic applied to sets of seeds (SSG), and the KL algorithm, we will refer to it as PHC/SSG+KL. It could easily be the case that the KL refinement is not worth the time it takes away from additional PHC iterations, so

	KL: 20 runs		X: 20 runs		% improvement over KL	
Graph	min	avg	min	avg	min	avg
test4	1,376	2,098.8	1,295	1,612.2	5.9	23.2
test5	2,257	4,393.8	2,138	2,606.3	5.3	40.7
test6	1,309	1,723.7	1,233	1,326.2	5.8	23.1
test2	1,274	1,512.7	1,281	1,357.4	-0.6	10.3
test3	1,147	2,829.1	1,013	1,693.1	11.7	40.2
19ks	1,461	2030.7	1,368	1,625.5	6.4	20.0
primary1	368	463.2	300	375.5	18.5	19.0
bm1	326	436.9	303	378.6	7.1	13.3
primary2	1,636	2,160.1	1,285	1,766.7	21.5	18.2

Table 1: Kernighan-Lin and Algorithm X: an empirical comparison. Algorithm X runs five times more slowly than the Kernighan-Lin (KL) algorithm.

we will also investigate the unadorned variant, PHC/SSG. Note that parallel hill-climbing could also be used with the single-vertex seed variant, just as a random generate-and-test strategy could be used here. We will also present results for these variants, notated as PHC/SG, PHC/SG+KL, RGT/SSG+KL, and RGT/SSG. Thus, we examine a total of 8 variants of the seed-growth strategy, with the intent not only of identifying the best technique for different kinds of graphs, but also of understanding the relative contributions of the different algorithmic elements.

3 Empirical Analysis

Heuristic algorithms for graph partitioning like the ones described above cannot be evaluated in a purely analytic fashion; empirical analysis is the only way to ascertain such an algorithm's utility. Unfortunately, empirical analysis of algorithm performance is often done poorly, which sometimes leads to erroneous conclusions. In the following subsection we discuss two common errors that are often committed in the empirical analysis of graph-partitioning algorithms. We then present empirical results for the seed-growth algorithms.

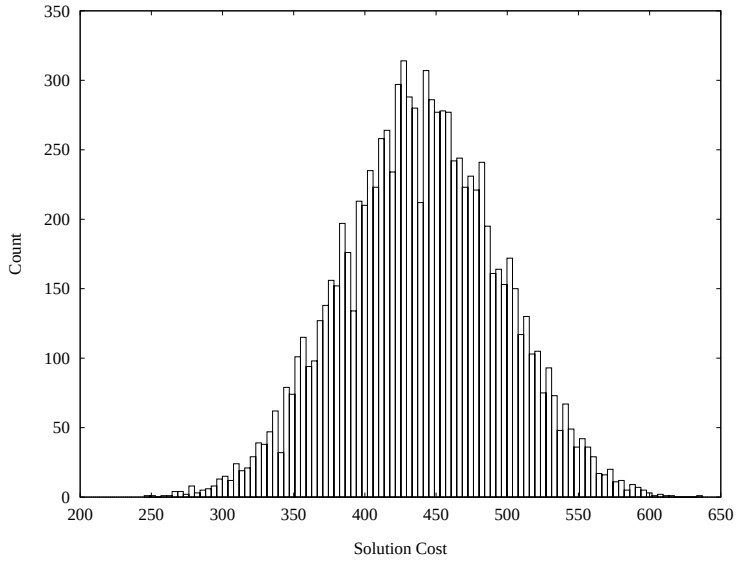

Figure 1: Histogram of solutions computed by the KL algorithm for graph `bm1`.

3.1 Caveats

Consider the evidence presented in Table 1. (This example is based on an empirical analysis reported by Wei and Cheng [27].) The table contains the average and minimum cut-set sizes of bisections for 9 different graphs, computed from 20 runs of the KL algorithm and 20 runs of some mystery Algorithm X. Although Algorithm X is five times more expensive than the KL algorithm, one might be tempted to conclude that the extra expense is indeed worthwhile, because its performance appears to be significantly better. However, the difference in performance is due solely to the extra time afforded Algorithm X, because Algorithm X merely returns the best of five runs of the KL algorithm! The moral is clear: Given the high variance of the distribution of results generated by the KL algorithm, any analysis that does not give equal time to KL will result in an inappropriate comparison.

The nature of the distribution of KL results provides a further opportunity for misleading analysis. Figure 1 shows the distribution of 10,000 values returned by the KL algorithm for graph `bm1`, which is derived from a benchmark VLSI circuit. Suppose that some hypothetical Algorithm Y generates a distribution of results with better mean but smaller variance: for instance, let us assume that it essentially always finds a bisection with

cut-set size between 300 and 350 for this graph. If one compares the best result from m runs of Algorithm Y with the best result from n runs of the KL algorithm to determine which algorithm is better (where m and n have been chosen to equate overall running times, of course), the answer one gets will be affected by the magnitude of n. 57 of the values in the histogram for KL are less than 300. Since the chance of obtaining a solution in the top $\frac{57}{10,000}$ from n samples is $1 - (1 - \frac{57}{10,000})^n$, n must be at least $\log_{0.9943} 0.5 \approx 122$ in order for KL to have at least a 50% chance of being declared the better algorithm by virtue of finding the best bisection. Therefore, if one can afford to wait the 15 seconds or so required for 400 runs of KL—as is typical for many applications involving graph partitioning—KL has a 90% chance of finding a better bisection than Algorithm Y and should be considered the better algorithm on the basis of the empirical evidence. When absolute performance is what matters most, several tens or even hundreds of runs of the KL algorithm may be required to do it justice; a statistical analysis of the distribution of results for a given graph can be used to estimate an appropriate minimum number of runs, if such an estimate is needed [26]. Conversely, any comparisons with KL that involve taking the best of as few as 10 or 20 runs—especially against algorithms with good average performance but low variance—would appear to be suspect, though such comparisons are not uncommon [6, 16, 27, 28].

3.2 Benchmarks

We investigated the performance of the seed-growth algorithms on the three most commonly used classes of benchmark problems: 18 graphs derived from benchmark VLSI circuit hypergraphs using the standard clique model,[1] 6 uniform random graphs, in which each possible edge is generated with fixed probability, and 6 geometric random graphs, in which vertices are randomly placed on a unit square and connected to all neighbors within a fixed radius. One would expect the geometric random graphs, but not the uniform ones, to exhibit exploitable structure [18]. Table 2 contains various statistics of the benchmark graphs. From left to right:

1. $|V|$, the number of vertices.
2. $|E|$, the number of edges.
3. $p(edge)$, the edge probability. This is the probability that a possible edge actually appears in the graph.

[1] The hypergraphs are freely available at `http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html`.

Graph	$\|V\|$	$\|E\|$	$p(edge)$	deg	$IQR(deg)$	$\overline{deg}$	$\sigma(deg)$	CC	Time
test4	1,515	126,349	0.11017	22	7–424	166.8	236.3	1	379.0
test5	2,595	274,173	0.08146	31	7–289	211.3	331.1	1	822.5
test6	1,752	122,449	0.07983	61	7–129	139.8	190.1	1	367.3
fract	149	869	0.07881	7	3–11	11.7	14.9	1	2.6
test2	1,663	105,441	0.07630	14	5–263	126.8	204.8	1	316.3
test3	1,607	65,442	0.05071	24	8–166	81.4	102.6	1	196.3
balu	801	15,470	0.04828	13	6–31	38.6	65.6	1	46.4
19ks	2,844	185,948	0.04600	18	4–201	130.8	201.5	1	557.8
primary1	833	6,257	0.01806	11	6–23	15.0	12.1	1	18.8
bm1	882	6,258	0.01611	10	5–20	14.2	12.2	1	18.8
primary2	3,014	37,252	0.00820	19	10–33	24.7	22.6	1	111.8
struct	1,952	8,542	0.00449	5	3–7	8.8	10.0	1	25.6
industry3	15,406	179,846	0.00152	11	7–20	23.3	50.8	45	539.5
s9234	5,866	16,003	0.00093	3	2–6	5.5	6.4	4	48.0
s13207	8,772	28,550	0.00074	2	2–5	6.5	11.0	7	85.7
s38584	20,995	143,590	0.00065	5	3–14	13.7	19.9	7	430.8
s15850	10,470	32,741	0.00060	3	2–6	6.3	8.2	16	98.2
s38417	23,949	86,645	0.00030	3	2–6	7.2	11.8	1	259.9
geo-rand-0.10	1,000	42,633	0.08535	90	71–101	85.3	20.5	1	127.9
geo-rand-0.08	1,000	36,134	0.07234	77	59–86	72.3	18.2	1	108.4
geo-rand-0.06	1,000	26,578	0.05321	54	46–62	53.2	12.0	1	79.7
dsj-geo-rand-0.04	1,000	18,015	0.03607	37	31–42	36.0	8.4	1	54.0
dsj-geo-rand-0.02	1,000	9,339	0.01870	19	15–22	18.7	5.2	1	28.0
dsj-geo-rand-0.01	1,000	4,696	0.00940	9	7–12	9.4	3.2	3	14.1
dsj-geo-rand-0.005	1,000	2,394	0.00479	5	3–6	4.8	2.2	23	7.2
unif-rand-0.10	1,000	49,614	0.09933	99	93–106	99.2	9.5	1	148.8
unif-rand-0.08	1,000	40,041	0.08016	80	74–86	80.1	8.6	1	120.1
unif-rand-0.06	1,000	29,865	0.05979	59	54–65	59.7	7.6	1	89.6
unif-rand-0.04	1,000	19,819	0.03968	39	36–43	39.6	5.8	1	59.5
dsj-unif-rand-0.02	1,000	10,107	0.02023	20	17–23	20.2	4.3	1	30.3
dsj-unif-rand-0.01	1,000	5,064	0.01014	10	8–12	10.1	3.1	1	15.2
dsj-unif-rand-0.005	1,000	2,496	0.00500	5	3–6	5.0	2.2	7	7.5
dsj-unif-rand-0.0025	1,000	1,272	0.00255	2	1–4	2.5	1.6	80	3.8

Table 2: Statistics of the benchmark graphs.

Graph	Best	Source
dsj-geo-rand-0.04	737	[18, Table XI]
dsj-geo-rand-0.02	222	[18, Table XI]
dsj-geo-rand-0.01	39	[18, Table XI]
dsj-geo-rand-0.005	1	[5, Table II]
dsj-unif-rand-0.02	3382	[5, Table II]
dsj-unif-rand-0.01	1362	[5, Table II]
dsj-unif-rand-0.005	445	[5, Table II]
dsj-unif-rand-0.0025	95	[5, Table II]

Table 3: Best known bisections of the Johnson et al. graphs.

4. $\tilde{deg}$, the median degree of a vertex in the graph.

5. $IQR(deg)$, the interquartile range of the degrees. In a sorted list of all vertex degrees, this statistic measures the spread between those degrees falling one-quarter and three-quarters of the way through the list.

6. $\overline{deg}$, the mean (or average) degree.

7. $\sigma(deg)$, the standard deviation of the vertex degrees from the mean.

8. CC, the number of connected components in the graph.

9. *Time*, measured in seconds on a DEC AlphaStation 500/500 (SPECint95 15.0, SPECfp95 20.4), allotted to each algorithm for finding a bisection of the graph. This does not include time needed to read the graph from disk, but does include time spent initializing algorithm-specific data structures. The time was determined by multiplying $|E|$ by 0.003, an arbitrary constant which was chosen to be large enough to give our implementation of the plain KL algorithm several hundred runs during every test. The running times range from 2.6 seconds for graph `fract` to 13.7 minutes for graph `test5`.

The names for the random graphs indicate their expected edge probabilities. Names prefixed with 'dsj' indicate standard benchmark graphs used by Johnson et al. [18] in their study of simulated annealing, the most thorough of previous empirical investigations of graph bisection. The lowest known cut-set sizes for these graphs appear in Table 3. Additional values of

p were chosen in order to match the range of edge probabilities in the circuit graphs, which we take as representative of important practical problems. To construct a geometric random graph with a desired edge probability p, we used a radius of $\sqrt{(|V|-1)p/(|V|\pi)}$.

3.3 Single-vertex Seed Sets

Table 4 presents an empirical comparison of the KL and RGT/SG algorithms. Here, and in later performance tables, we report the following statistics for each graph in our test suite:

1. *Number of KL runs:* The average number of independent runs of the KL algorithm that were completed within the allotted time.
2. *Mean minimum cut-set size for KL:* The average minimum cut-set size found over 25 tests of k runs each, where k is the number of runs completed within the time limit.
3. *Standard deviation of minimum cut-set size for KL:* The standard deviation of the minimum cut-set size found over the 25 tests.
4. *Median minimum cut-set size for KL:* The median best solution from the 25 tests.
5. *Number of iterations of RGT/SG:* The average number of iterations of the search strategy. In this case, this is the number of independent random SG solutions generated.
6. *Average minimum cut-set size for RGT/SG:* The average minimum cut-set size found over 25 runs of the RGT/SG algorithm.
7. *Standard deviation of minimum cut-set size for RGT/SG:* The standard deviation of the minimum cut-set sizes found over the 25 tests.
8. *Median minimum cut-set size for KL:* The median best solution from the 25 tests.
9. *Improvement over KL:* The average improvement of the RGT/SG algorithm over the KL algorithm, expressed as a percentage of the average minimum cut-set size for KL.

Overall, the results are encouraging, but mixed. RGT/SG shows enormous improvements on most of the highly structured VLSI graphs and poor

	KL				RGT/SG				
Graph	Runs	Mean	σ	Med.	Iters	Mean	σ	Med.	Impr.
test4	868.4	1256.6	10.4	1,254	2753.0	1271.8	0.4	1,272	-1.2
test5	792.5	2034.3	26.5	2,035	2999.3	2051.0	6.3	2,049	-0.8
test6	982.6	1202.9	8.4	1,200	3125.4	1187.0	1.5	1,187	1.3
fract	573.9	55.0	0.0	55	2055.9	55.0	0.0	55	0.0
test2	838.9	1239.7	13.9	1,240	3049.6	1262.8	7.9	1,262	-1.9
test3	914.3	899.3	25.9	898	2796.3	839.0	4.1	839	6.7
balu	818.0	584.5	0.6	584	2705.0	586.0	0.2	586	-0.3
19ks	812.0	1167.2	59.4	1,171	3162.7	986.4	1.2	986	15.5
primary1	576.2	277.5	15.9	279	2341.4	233.3	1.8	232	15.9
bm1	563.9	269.2	19.2	271	2465.9	227.4	1.0	228	15.5
primary2	685.6	1299.6	90.0	1,296	2570.3	624.6	1.2	625	51.9
struct	301.7	359.6	14.5	358	1958.4	324.4	4.1	322	9.8
industry3	416.5	6697.6	307.3	6,760	2588.9	962.7	43.0	978	85.6
s9234	216.2	664.6	29.1	669	1629.6	206.3	10.7	204	69.0
s13207	228.3	800.2	34.7	802	1764.3	152.0	5.9	155	81.0
s38584	285.2	3545.8	154.6	3,568	2204.3	478.1	31.1	490	86.5
s15850	196.2	986.7	46.8	998	1787.5	219.7	14.0	213	77.7
s38417	186.6	2270.5	91.0	2,274	1771.3	351.8	12.4	353	84.5
geo-rand-0.10	934.8	3094.0	0.0	3,094	2932.7	3094.0	0.0	3,094	0.0
geo-rand-0.08	972.5	2041.0	0.0	2,041	2862.7	2041.0	0.0	2,041	0.0
geo-rand-0.06	787.2	1274.0	0.0	1,274	2899.3	1276.0	0.0	1,276	-0.2
dsj-geo-rand-0.04	646.1	737.0	0.0	737	3260.3	740.3	1.1	740	-0.4
dsj-geo-rand-0.02	461.2	222.0	0.0	222	2902.2	239.4	0.5	239	-7.9
dsj-geo-rand-0.01	335.4	55.8	5.2	55	2407.8	43.6	0.5	44	21.7
dsj-geo-rand-0.005	235.9	32.1	4.3	31	1724.0	1.0	0.0	1	96.9
unif-rand-0.10	446.6	21306.8	13.6	21,303	3082.6	21479.0	16.3	21,477	-0.8
unif-rand-0.08	437.2	16827.7	12.7	16,828	3014.2	16981.3	11.2	16,981	-0.9
unif-rand-0.06	437.1	12177.3	10.8	12,176	2859.0	12302.2	11.4	12,299	-1.0
unif-rand-0.04	421.4	7618.9	10.4	7,619	2772.2	7730.4	8.8	7,731	-1.5
dsj-unif-rand-0.02	374.5	3410.2	6.6	3,409	2929.1	3476.5	8.1	3,476	-1.9
dsj-unif-rand-0.01	313.4	1387.6	4.2	1,388	2459.7	1429.3	2.4	1,429	-3.0
dsj-unif-rand-0.005	223.8	470.2	3.6	469	1784.1	488.3	3.9	488	-3.8
dsj-unif-rand-0.0025	122.9	108.3	1.8	108	1220.0	116.2	1.5	116	-7.3

Table 4: The performance of Kernighan-Lin and RGT/SG.

performance on random graphs. Edge probability seems to be a better predictor of performance on the VLSI graphs than mean degree. Interestingly, the relative quality of seed-growth solutions falls as the density of the random graphs decreases, and then exhibits a sharp reversal for the geometric random graphs as the edge probability passes from 0.019 to 0.009. The sparsest geometric graph tested showed the largest advantage for RGT/SG: the seed-growth algorithm consistently found a bisection cutting only a single edge while the median KL solution cut 31 edges. The close correspondence between RGT/SG's relative performance on the VLSI and geometric random graphs is the first verification we know of that the random geometric graph model is useful for modeling algorithm behavior on real-world problems.

The results for RGT/SG may appear ordinary relative to the results that have been reported recently for various partitioning heuristics in the CAD literature. It is important to note, however, that a direct comparison using the VLSI graphs is inappropriate in many cases, as results for many other algorithms are reported in terms of hypergraph nets. Also, many heuristics address variations of the bisection problem in which imbalance is tolerated and vertices have associated weights. Most importantly, recall the large number of KL runs we use, over 400 on average. Note that comparing the 'Min' columns in Table 1 shows the improvement one can get by taking the best of 100 runs of the KL algorithm versus the best of 20 runs; moreover, the best of 400 runs is quite an improvement, on average, over the best of 100 runs. Thus, our time-equated results cannot be directly compared to previously published improvements. In our appendix, we include approximate comparisons to published results on the random graphs. But first, we continue our investigation of methods based on seed-growth.

3.4 KL as a postprocess

Since the seed-growth heuristic adds vertices in a greedy manner and never reconsiders past decisions, it seems reasonable to expect that the resulting solutions could contain flaws that would be easily recognized and corrected by a swap-based algorithm such as KL. What is not clear is whether the extra time consumed by such a refinement postprocess would be better spent considering additional seed-growth solutions.

Table 5 presents our empirical answer to this question. The response is solidly affirmative, with RGT/SG+KL outperforming both KL and RGT/SG on every graph, even though it completes from one-third to one-sixth as many iterations as RGT/SG. In a reversal from RGT/SG, the advantage over KL increases with graph sparsity for both classes of random graphs.

	KL				RGT/SG+KL				
Graph	Runs	Mean	σ	Med.	Iters	Mean	σ	Med.	Impr.
test4	868.4	1256.6	10.4	1,254	954.3	1246.4	6.5	1,249	0.8
test5	792.5	2034.3	26.5	2,035	948.4	1957.6	10.4	1,960	3.8
test6	982.6	1202.9	8.4	1,200	969.0	1184.4	0.5	1,184	1.5
fract	573.9	55.0	0.0	55	729.7	55.0	0.0	55	0.0
test2	838.9	1239.7	13.9	1,240	835.2	1230.7	7.7	1,233	0.7
test3	914.3	899.3	25.9	898	988.4	829.2	3.8	827	7.8
balu	818.0	584.5	0.6	584	826.1	584.0	0.0	584	0.1
19ks	812.0	1167.2	59.4	1,171	973.2	969.9	1.8	970	16.9
primary1	576.2	277.5	15.9	279	761.6	220.8	1.7	220	20.4
bm1	563.9	269.2	19.2	271	705.0	218.6	0.9	219	18.8
primary2	685.6	1299.6	90.0	1,296	834.0	596.1	5.2	599	54.1
struct	301.7	359.6	14.5	358	397.5	320.8	3.5	322	10.8
industry3	416.5	6697.6	307.3	6,760	732.0	880.1	14.9	889	86.9
s9234	216.2	664.6	29.1	669	403.1	193.7	16.4	194	70.9
s13207	228.3	800.2	34.7	802	429.8	149.8	7.5	150	81.3
s38584	285.2	3545.8	154.6	3,568	564.7	402.3	13.3	399	88.7
s15850	196.2	986.7	46.8	998	446.4	198.6	16.1	198	79.9
s38417	186.6	2270.5	91.0	2,274	308.4	270.5	7.4	266	88.1
geo-rand-0.10	934.8	3094.0	0.0	3,094	826.1	3094.0	0.0	3,094	0.0
geo-rand-0.08	972.5	2041.0	0.0	2,041	859.4	2041.0	0.0	2,041	0.0
geo-rand-0.06	787.2	1274.0	0.0	1,274	786.4	1274.0	0.0	1,274	0.0
dsj-geo-rand-0.04	646.1	737.0	0.0	737	800.4	737.0	0.0	737	0.0
dsj-geo-rand-0.02	461.2	222.0	0.0	222	648.8	222.0	0.0	222	0.0
dsj-geo-rand-0.01	335.4	55.8	5.2	55	638.1	39.4	0.5	39	29.3
dsj-geo-rand-0.005	235.9	32.1	4.3	31	548.0	1.0	0.0	1	96.9
unif-rand-0.10	446.6	21306.8	13.6	21,303	458.9	21301.5	12.8	21,298	0.0
unif-rand-0.08	437.2	16827.7	12.7	16,828	463.5	16818.2	14.2	16,816	0.1
unif-rand-0.06	437.1	12177.3	10.8	12,176	458.8	12168.3	11.9	12,170	0.1
unif-rand-0.04	421.4	7618.9	10.4	7,619	442.7	7614.3	8.0	7,614	0.1
dsj-unif-rand-0.02	374.5	3410.2	6.6	3,409	443.1	3403.0	5.8	3,404	0.2
dsj-unif-rand-0.01	313.4	1387.6	4.2	1,388	385.9	1383.2	4.1	1,384	0.3
dsj-unif-rand-0.005	223.8	470.2	3.6	469	302.6	466.4	2.6	466	0.8
dsj-unif-rand-0.0025	122.9	108.3	1.8	108	203.4	107.4	1.2	107	0.9

Table 5: The performance of KL and RGT/SG+KL.

We hypothesize that the purely local connectivity used by the seed-growth heuristic provides less information on sparser graphs, providing further opportunities for clean-up by the KL algorithm.

3.5 Parallel Hill-Climbing

As we mentioned above, random generate-and-test can be viewed as a simplistic search strategy, in which one attempts to find initial seed vertices that can be greedily grown into a good bisection. It might be the case, then, that a more sophisticated search strategy would be more effective in identifying promising seeds. To test this hypothesis, we used the parallel hill-climbing technique described above in Section 2.2 as a replacement for random generate-and-test.

In experiments analogous to the ones described above, we compared PHC/SG and PHC/SG+KL and determined that KL refinement is uniformly beneficial. Therefore, in Table 6 we present a comparison of PHC/SG+KL with RGT/SG+KL. For algorithms incorporating PHC, the 'Iters' column does not include solutions generated to initialize the population (although of course this phase of the algorithm is included in the timing measurements). The PHC version is able to complete many more iterations than the RGT algorithm because of its selective use of the expensive KL refinement. The two algorithms perform very similarly, although the more sophisticated search technique yields a slight advantage for the sparse VLSI graphs.

3.6 Larger Seed Sets

It seems reasonable to conjecture that seeding more than one vertex in each partition could provide benefits, particularly in conjunction with a search procedure. First, vertices that would be improperly assigned by the greedy growth heuristic could be correctly assigned from the start. It is unclear, however, how often this would lead to the incorrect assignment of other neighboring vertices. Second, assigning multiple seeds could implicitly capture clustering inherent in structured graphs. If a vertex in a cluster were part of a seed set, this makes it likely that the entire cluster will be assigned to that vertex's partition. Searching over possible seed sets could thereby function as an implicit search over clusters.

We investigated the performance of the seed growth heuristic with *size* now set to $0.01|V|$ instead of 1. As mentioned previously, we refer to this heuristic as SSG, since we are now growing from sets of seeds in each partition. KL refinement was found to be worthwhile for seed-set solutions

	RGT/SG+KL				PHC/SG+KL				
Graph	Iters	Mean	σ	Med.	Iters	Mean	σ	Med.	Impr.
test4	954.3	1246.4	6.5	1,249	2587.5	1250.3	3.4	1,251	-0.3
test5	948.4	1957.6	10.4	1,960	2764.3	1963.3	4.0	1,960	-0.3
test6	969.0	1184.4	0.5	1,184	2208.1	1185.0	0.5	1,185	-0.1
fract	729.7	55.0	0.0	55	1645.8	55.0	0.0	55	0.0
test2	835.2	1230.7	7.7	1,233	2795.5	1244.8	10.3	1,250	-1.1
test3	988.4	829.2	3.8	827	2688.6	827.3	1.4	827	0.2
balu	826.1	584.0	0.0	584	2253.4	584.8	0.4	585	-0.1
19ks	973.2	969.9	1.8	970	2669.7	970.1	1.9	971	-0.0
primary1	761.6	220.8	1.7	220	2176.4	220.0	0.2	220	0.4
bm1	705.0	218.6	0.9	219	2181.6	217.8	1.9	218	0.3
primary2	834.0	596.1	5.2	599	2155.2	587.4	6.6	582	1.5
struct	397.5	320.8	3.5	322	1581.8	321.6	1.2	322	-0.2
industry3	732.0	880.1	14.9	889	2333.9	849.2	35.7	857	3.5
s9234	403.1	193.7	16.4	194	1373.1	178.3	9.9	178	7.9
s13207	429.8	149.8	7.5	150	1421.9	146.8	4.9	149	2.0
s38584	564.7	402.3	13.3	399	1880.7	387.2	6.8	388	3.7
s15850	446.4	198.6	16.1	198	1522.4	181.5	8.1	182	8.6
s38417	308.4	270.5	7.4	266	1321.8	267.4	4.6	266	1.1
geo-rand-0.10	826.1	3094.0	0.0	3,094	2952.3	3094.0	0.0	3,094	0.0
geo-rand-0.08	859.4	2041.0	0.0	2,041	2816.7	2041.0	0.0	2,041	0.0
geo-rand-0.06	786.4	1274.0	0.0	1,274	2657.6	1274.0	0.0	1,274	0.0
dsj-geo-rand-0.04	800.4	737.0	0.0	737	2728.0	737.0	0.0	737	0.0
dsj-geo-rand-0.02	648.8	222.0	0.0	222	1949.0	224.0	4.1	222	-0.9
dsj-geo-rand-0.01	638.1	39.4	0.5	39	1976.3	39.8	0.4	40	-0.9
dsj-geo-rand-0.005	548.0	1.0	0.0	1	1380.1	1.0	0.0	1	0.0
unif-rand-0.10	458.9	21301.5	12.8	21,298	2751.0	21304.6	12.6	21,307	-0.0
unif-rand-0.08	463.5	16818.2	14.2	16,816	2762.0	16818.7	9.7	16,819	-0.0
unif-rand-0.06	458.8	12168.3	11.9	12,170	2726.7	12171.9	11.4	12,172	-0.0
unif-rand-0.04	442.7	7614.3	8.0	7,614	2634.7	7611.0	9.1	7,611	0.0
dsj-unif-rand-0.02	443.1	3403.0	5.8	3,404	2356.0	3400.8	7.4	3,398	0.1
dsj-unif-rand-0.01	385.9	1383.2	4.1	1,384	1914.3	1382.6	4.9	1,382	0.0
dsj-unif-rand-0.005	302.6	466.4	2.6	466	1441.7	465.7	1.9	466	0.2
dsj-unif-rand-0.0025	203.4	107.4	1.2	107	860.9	107.0	2.1	108	0.3

Table 6: The performance of RGT/SG+KL and PHC/SG+KL.

	RGT/SG+KL				RGT/SSG+KL				
Graph	Iters	Mean	σ	Med.	Iters	Mean	σ	Med.	Impr.
test4	954.3	1246.4	6.5	1,249	935.0	1233.3	3.7	1,233	1.0
test5	948.4	1957.6	10.4	1,960	1019.5	1943.5	15.6	1,947	0.7
test6	969.0	1184.4	0.5	1,184	1023.5	1184.9	1.0	1,185	-0.0
fract	729.7	55.0	0.0	55	754.8	55.0	0.0	55	0.0
test2	835.2	1230.7	7.7	1,233	872.0	1212.9	5.1	1,212	1.4
test3	988.4	829.2	3.8	827	953.5	829.4	2.1	829	-0.0
balu	826.1	584.0	0.0	584	794.7	584.0	0.0	584	0.0
19ks	973.2	969.9	1.8	970	953.6	969.2	4.0	969	0.1
primary1	761.6	220.8	1.7	220	737.0	219.6	1.9	219	0.6
bm1	705.0	218.6	0.9	219	702.1	214.5	2.8	214	1.8
primary2	834.0	596.1	5.2	599	812.2	587.2	18.0	585	1.5
struct	397.5	320.8	3.5	322	387.3	332.2	8.5	334	-3.6
industry3	732.0	880.1	14.9	889	692.5	909.1	86.1	880	-3.3
s9234	403.1	193.7	16.4	194	321.5	201.7	18.2	197	-4.1
s13207	429.8	149.8	7.5	150	359.1	215.3	22.8	215	-43.7
s38584	564.7	402.3	13.3	399	463.8	525.1	43.1	522	-30.5
s15850	446.4	198.6	16.1	198	328.9	268.3	19.6	268	-35.1
s38417	308.4	270.5	7.4	266	256.1	621.6	51.2	616	-129.8
geo-rand-0.10	826.1	3094.0	0.0	3,094	866.9	3094.0	0.0	3,094	0.0
geo-rand-0.08	859.4	2041.0	0.0	2,041	874.9	2041.0	0.0	2,041	0.0
geo-rand-0.06	786.4	1274.0	0.0	1,274	792.5	1274.0	0.0	1,274	0.0
dsj-geo-rand-0.04	800.4	737.0	0.0	737	757.3	737.0	0.0	737	0.0
dsj-geo-rand-0.02	648.8	222.0	0.0	222	630.0	222.0	0.0	222	0.0
dsj-geo-rand-0.01	638.1	39.4	0.5	39	591.0	39.1	0.3	39	0.8
dsj-geo-rand-0.005	548.0	1.0	0.0	1	449.9	1.0	0.0	1	0.0
unif-rand-0.10	458.9	21301.5	12.8	21,298	468.2	21303.4	11.3	21,305	-0.0
unif-rand-0.08	463.5	16818.2	14.2	16,816	448.4	16817.0	13.1	16,821	0.0
unif-rand-0.06	458.8	12168.3	11.9	12,170	454.5	12173.7	8.9	12,176	-0.0
unif-rand-0.04	442.7	7614.3	8.0	7,614	436.3	7610.3	10.6	7,608	0.1
dsj-unif-rand-0.02	443.1	3403.0	5.8	3,404	428.3	3404.0	6.8	3,405	-0.0
dsj-unif-rand-0.01	385.9	1383.2	4.1	1,384	368.5	1383.3	4.2	1,383	-0.0
dsj-unif-rand-0.005	302.6	466.4	2.6	466	278.4	467.0	3.5	467	-0.1
dsj-unif-rand-0.0025	203.4	107.4	1.2	107	180.9	107.5	1.0	107	-0.1

Table 7: The performance of RGT/SG+KL and RGT/SSG+KL.

	PHC/SG+KL				PHC/SSG+KL				
Graph	Iters	Mean	σ	Med.	Iters	Mean	σ	Med.	Impr.
test4	2587.5	1250.3	3.4	1,251	2288.5	1241.2	8.1	1,240	0.7
test5	2764.3	1963.3	4.0	1,960	2100.9	1951.6	7.2	1,952	0.6
test6	2208.1	1185.0	0.5	1,185	2098.8	1187.2	2.7	1,186	-0.2
fract	1645.8	55.0	0.0	55	1699.4	55.0	0.0	55	0.0
test2	2795.5	1244.8	10.3	1,250	2167.1	1235.3	17.4	1,238	0.8
test3	2688.6	827.3	1.4	827	2275.4	827.4	0.9	827	-0.0
balu	2253.4	584.8	0.4	585	1959.6	584.0	0.2	584	0.1
19ks	2669.7	970.1	1.9	971	1947.0	978.4	21.3	972	-0.9
primary1	2176.4	220.0	0.2	220	1858.5	218.1	1.5	218	0.9
bm1	2181.6	217.8	1.9	218	1814.4	212.2	4.4	213	2.6
primary2	2155.2	587.4	6.6	582	1692.1	583.9	26.9	579	0.6
struct	1581.8	321.6	1.2	322	1220.0	329.4	6.6	328	-2.4
industry3	2333.9	849.2	35.7	857	1424.6	981.0	131.4	972	-15.5
s9234	1373.1	178.3	9.9	178	650.8	187.5	18.9	184	-5.1
s13207	1421.9	146.8	4.9	149	684.9	201.3	29.2	204	-37.1
s38584	1880.7	387.2	6.8	388	904.9	564.9	68.2	576	-45.9
s15850	1522.4	181.5	8.1	182	656.0	254.1	31.8	257	-40.0
s38417	1321.8	267.4	4.6	266	552.8	564.0	67.5	572	-110.9
geo-rand-0.10	2952.3	3094.0	0.0	3,094	2100.6	3094.0	0.0	3,094	0.0
geo-rand-0.08	2816.7	2041.0	0.0	2,041	2054.5	2041.0	0.0	2,041	0.0
geo-rand-0.06	2657.6	1274.0	0.0	1,274	1871.8	1274.0	0.0	1,274	0.0
dsj-geo-rand-0.04	2728.0	737.0	0.0	737	2068.1	737.0	0.0	737	0.0
dsj-geo-rand-0.02	1949.0	224.0	4.1	222	1866.7	222.0	0.0	222	0.9
dsj-geo-rand-0.01	1976.3	39.8	0.4	40	1528.9	39.1	0.3	39	1.6
dsj-geo-rand-0.005	1380.1	1.0	0.0	1	1017.9	1.0	0.2	1	-4.0
unif-rand-0.10	2751.0	21304.6	12.6	21,307	2208.6	21295.9	12.2	21,294	0.0
unif-rand-0.08	2762.0	16818.7	9.7	16,819	2186.4	16811.3	12.8	16,814	0.0
unif-rand-0.06	2726.7	12171.9	11.4	12,172	2090.0	12159.5	11.8	12,157	0.1
unif-rand-0.04	2634.7	7611.0	9.1	7,611	1977.0	7607.6	8.4	7,605	0.0
dsj-unif-rand-0.02	2356.0	3400.8	7.4	3,398	2096.5	3395.8	4.9	3,395	0.1
dsj-unif-rand-0.01	1914.3	1382.6	4.9	1,382	1788.2	1382.7	4.7	1,383	-0.0
dsj-unif-rand-0.005	1441.7	465.7	1.9	466	1244.5	464.1	3.2	463	0.4
dsj-unif-rand-0.0025	860.9	107.0	2.1	108	610.3	106.0	1.7	106	0.9

Table 8: The performance of PHC/SG+KL and PHC/SSG+KL.

discovered using either random generate-and-test or parallel hill-climbing. Therefore, in Table 7 we compare RGT/SG+KL with its set-oriented cousin, RGT/SSG+KL. The seed-set version exhibits similar performance, except for the the sparse VLSI graphs, on which the single-vertex algorithm is dramatically superior. Similar behavior was evident for the PHC versions, whose performance is shown in Table 8. The use of PHC instead of RGT seemed to improve performance slightly on sparse VLSI graphs, which was the same effect we saw with single-vertex seeds. (When not using KL refinement, PHC/SSG consistently provided an advantage over RGT/SSG, with the advantage increasing with sparsity on random graphs.) Overall, it appears that any implicit clustering effect does not make up for the extra constraints placed on the greedy seed-growth heuristic.

4 Conclusions

We have investigated several algorithms for graph bisection that are based on a simple seed-growth heuristic. Through time-equated experiments on three families of benchmark graphs, we have disambiguated the various contributions of seed-set size, search technique, and KL refinement. Algorithms incorporating the seed-growth heuristic consistently outperform KL. One of the novel combinations we considered, PHC/SG+KL, emerged as the champion, although RGT/SG+KL is just as good for uniform random graphs.

One danger of time-equated comparisons is that an algorithm that cannot make efficient use of extra time may be displayed at a disadvantage when compared to an algorithm that depends on long running times to become competitive. To ensure that the advantage displayed by PHC/SG+KL over KL is not due to this phenomenon, we present results in Table 9 that were compiled after each algorithm had run for one-seventh of our usual computation time. PHC/SG+KL's advantage remains clear.

The PHC/SG+KL algorithm is certainly an improvement over the KL algorithm, but it remains to be seen how effective it is relative to other recently reported algorithms that use explicit clustering heuristics. The experimental methodology and large-sample KL results we have presented here can form the basis of such investigations.

In an appendix, we report some approximate comparisons to published results on the random graphs of Johnson et al., although only broad conclusions can be drawn. It appears that combining seed-growth with a refinement technique with similar strengths, such as KL, is not as effective as attempting to boost its performance on more uniform graphs with a post-

	KL				PHC/SG+KL				
Graph	Runs	Mean	σ	Med.	Iters	Mean	σ	Med.	Impr.
test4	124.5	1281.2	19.7	1,280	218.1	1252.4	1.9	1,252	2.3
test5	118.9	2098.8	40.3	2,101	238.5	1966.3	5.2	1,967	6.3
test6	140.1	1224.2	18.6	1,222	229.4	1185.0	0.5	1,185	3.2
fract	76.1	55.0	0.0	55	92.5	55.0	0.0	55	0.0
test2	119.5	1262.0	19.5	1,261	235.6	1252.0	7.1	1,251	0.8
test3	130.5	927.9	40.9	927	229.5	837.2	9.0	837	9.8
balu	116.9	585.9	1.0	586	204.5	584.8	0.4	585	0.2
19ks	118.3	1286.8	64.6	1,280	240.3	973.1	2.6	973	24.4
primary1	81.5	310.9	22.5	312	152.1	224.0	2.6	224	28.0
bm1	79.7	299.7	24.5	300	152.1	219.0	1.8	219	26.9
primary2	96.8	1489.6	131.4	1,538	165.1	597.1	10.2	600	59.9
struct	42.9	398.4	29.3	396	66.5	330.1	5.4	330	17.1
industry3	59.3	7286.5	324.3	7,265	160.1	898.0	37.9	889	87.7
s9234	30.8	701.4	26.7	700	33.1	212.4	23.8	209	69.7
s13207	32.6	860.7	46.4	865	44.6	162.7	15.4	150	81.1
s38584	40.8	3836.4	192.2	3,861	104.9	429.5	42.0	414	88.8
s15850	28.0	1044.5	62.1	1,044	51.9	212.4	20.7	209	79.7
s38417	26.7	2432.0	97.0	2,422	18.7	276.0	11.3	274	88.7
geo-rand-0.10	133.1	3094.0	0.0	3,094	252.7	3094.0	0.0	3,094	0.0
geo-rand-0.08	138.1	2041.0	0.0	2,041	273.4	2041.0	0.0	2,041	0.0
geo-rand-0.06	113.0	1274.0	0.0	1,274	230.6	1274.0	0.0	1,274	0.0
dsj-geo-rand-0.04	91.4	737.0	0.0	737	239.6	737.0	0.0	737	0.0
dsj-geo-rand-0.02	64.9	225.4	5.1	222	174.9	224.7	4.5	222	0.3
dsj-geo-rand-0.01	46.7	70.2	10.9	68	118.9	39.8	0.4	40	43.3
dsj-geo-rand-0.005	32.8	40.1	7.2	41	73.1	1.2	0.4	1	97.1
unif-rand-0.10	63.5	21332.7	21.9	21,337	147.5	21322.7	15.2	21,324	0.0
unif-rand-0.08	62.3	16850.0	14.4	16,849	146.4	16831.4	15.8	16,830	0.1
unif-rand-0.06	61.6	12197.5	13.5	12,194	146.4	12185.7	12.1	12,186	0.1
unif-rand-0.04	59.1	7636.5	15.7	7,638	141.7	7626.5	11.5	7,626	0.1
dsj-unif-rand-0.02	51.7	3423.1	10.5	3,422	110.1	3409.7	10.0	3,411	0.4
dsj-unif-rand-0.01	43.9	1396.6	4.8	1,395	74.5	1390.2	5.1	1,390	0.5
dsj-unif-rand-0.005	31.3	475.4	3.8	475	24.2	471.5	3.4	472	0.8
dsj-unif-rand-0.0025	17.5	112.6	2.7	113	0.0	110.7	2.2	110	1.7

Table 9: The performance of KL and PHC/SG+KL when both are run for one seventh of our usual computation time.

process closer to simulated annealing.

Our agenda for future work includes a thorough time-equated empirical comparison of the most promising clustering-based heuristics for graph bisection, including PHC/SG+KL, and an attempt to discover further correlates between quantitative measures of a graph's structure and the performance of seed-growth heuristics.

Furthermore, we plan to generalize the PHC/SG+KL algorithm to other graph-partitioning problems. In commonly encountered problems of practical significance, more than two partitions are permitted, the requirement of exact equality of partition sizes is relaxed, and the vertices and edges are weighted. The simple nature of the seed-growth heuristic should allow for straightforward generalization to these cases.

5 Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant Nos. IRI-9350192, and IRI-9618848. We also thank David S. Johnson, Jason Cong, M'Lissa Smith, and Chuck Alpert for generously providing benchmark circuits and graphs, and Sandy Staff for help in compiling our bibliography and obtaining reprints.

A Comparisons with Other Algorithms

In this appendix, we broaden our perspective on the seed-growth algorithms we have considered in this paper by comparing them to algorithms other than KL. First we present results from our implementation of a simple spectral bisection algorithm, then we present an approximate comparison of PHC/SG+KL and RGT/SG+KL with published results from the literature.

A.1 Spectral Bisection

Although KL is the most popular benchmark algorithm for graph bisection, it could be regarded as part of the same algorithmic approach to graph bisection as seed-growth, since they both consider moving vertices based on edge crossings. We also implemented another standard technique that operates on a very different principle. This method, known as spectral bisection [1, 17, 15], uses an eigenvector computation to embed a graph along a line such that the sum of the squared distances between embedded connected vertices is minimized. Then we join the ends of the line to form a circle, and

consider each of the $|V|/2$ bisections resulting from breaking the ring into two equal halves, keeping contiguous vertices together. To take advantage of additional computation time, we can incrementally compute additional eigenvectors and their associated embeddings. More sophisticated variants of spectral bisection can be constructed by combining multiple embeddings [10].

Table 10 compares KL with spectral bisection. Unfortunately, even though we used a popular implementation of the Lanczos algorithm for sparse matrices,[2] the eigenvector computation takes $O(|V|^{1.4})$ time in the worst case, and occasionally took longer than the time allotted for a particular graph. Such graphs are not listed. This was especially a problem for graphs with many components, since the embedding technique needs eigenvector $c+1$ for a graph with c components. We also had difficulty reliably extracting many additional eigenvectors for some graphs, due to loss of orthogonality during the eigenvector computation. This prevented the spectral methods from using the full allotted computation time on those graphs. To permit a time-equated comparison against KL, we have listed results obtained after only 55% of our usual amount of computation time had elapsed. (In the trade-off between waiting for the first embedding of some graphs and being unable to generate additional embeddings for others, this value allowed us to present the most results.) The average number of eigenvector embeddings considered is listed in the 'Vecs' column. This varies because of fluctuations in the convergence times of the eigenvector code. Variation in solution quality can be attributed to the variation in the number of embeddings considered, as well as numerical inaccuracies past the fifth decimal place.

The spectral method performs surprisingly poorly against KL, particularly on sparse graphs. It finds better bisections for `primary2` and `19ks` than KL can, even when KL is given the usual time limit, although not as good as those found in an equal amount of time by the seed-growth algorithms we have considered (PHC/SG+KL has a time-equated mean cut-set size of 588.1 on `primary2` and 970.5 on `19ks`).

We saw with seed-growth algorithms that the increase in solution quality afforded by KL refinement can be well worth the time spent. In Table 11 we see that a KL refinement postprocess also improves the spectral solutions, although the improvement is not enough to provide a consistent advantage over KL alone. Spectral+KL outperforms KL only on the sparser VLSI

[2]The LASO package by D. S. Scott and B. N. Parlett, freely available at `http://www.netlib.org/laso`.

	KL				Spectral Bisection				
Graph	Runs	Mean	σ	Med.	Vecs.	Mean	σ	Med.	Impr.
`test4`	478.2	1260.9	12.9	1,258	4.4	1323.0	0.0	1,323	-4.9
`test6`	539.9	1209.3	13.2	1,206	4.7	1388.5	0.9	1,389	-14.8
`fract`	311.1	55.0	0.0	55	6.5	73.0	0.0	73	-32.7
`test2`	461.7	1247.2	18.3	1,249	3.7	1523.0	0.0	1,523	-22.1
`test3`	503.2	905.0	24.6	909	4.1	1288.0	0.0	1,288	-42.3
`balu`	450.6	584.8	0.7	585	6.9	809.0	0.0	809	-38.3
`19ks`	447.7	1201.8	60.6	1,203	2.3	1080.9	14.3	1,085	10.1
`primary1`	316.5	288.3	16.6	289	7.1	312.5	4.5	314	-8.4
`bm1`	309.6	280.4	20.0	284	6.1	301.0	0.0	301	-7.3
`primary2`	376.0	1332.2	89.4	1,326	5.5	947.8	0.6	948	28.9
`struct`	165.7	371.4	17.9	370	6.5	401.0	0.0	401	-8.0
geo-rand-0.10	514.0	3094.0	0.0	3,094	29.1	3127.0	0.0	3,127	-1.1
geo-rand-0.08	533.2	2041.0	0.0	2,041	29.8	2068.0	0.0	2,068	-1.3
geo-rand-0.06	433.5	1274.0	0.0	1,274	29.5	1298.0	0.0	1,298	-1.9
dsj-geo-rand-0.04	355.5	737.0	0.0	737	25.3	1062.0	0.0	1,062	-44.1
dsj-geo-rand-0.02	253.8	222.0	0.0	222	15.9	297.0	0.0	297	-33.8
dsj-geo-rand-0.01	184.1	57.7	6.4	56	4.5	74.0	0.0	74	-28.2
unif-rand-0.10	246.2	21316.6	18.3	21,318	23.9	22525.0	0.0	22,525	-5.7
unif-rand-0.08	240.8	16834.6	16.1	16,835	21.9	17935.0	0.0	17,935	-6.5
unif-rand-0.06	240.2	12182.3	13.7	12,180	22.6	13216.4	3.1	13,217	-8.5
unif-rand-0.04	230.4	7623.4	13.4	7,625	20.6	8362.0	0.0	8,362	-9.7
dsj-unif-rand-0.02	204.4	3412.8	8.5	3,411	18.6	3907.0	0.0	3,907	-14.5
dsj-unif-rand-0.01	171.4	1389.8	4.2	1,390	12.6	1685.0	0.0	1,685	-21.2

Table 10: The performance of Kernighan-Lin and spectral bisection when both are given 55% of our usual computation time.

	KL				Spectral+KL				
Graph	Runs	Mean	σ	Med.	Vecs.	Mean	σ	Med.	Impr.
`test4`	478.2	1260.9	12.9	1,258	4.1	1295.0	0.0	1,295	-2.7
`test6`	539.9	1209.3	13.2	1,206	4.2	1205.1	0.0	1,205	0.4
`fract`	311.1	55.0	0.0	55	6.0	55.0	0.0	55	0.0
`test2`	461.7	1247.2	18.3	1,249	3.0	1378.0	0.0	1,378	-10.5
`test3`	503.2	905.0	24.6	909	3.9	908.3	0.3	908	-0.4
`balu`	450.6	584.8	0.7	585	6.9	587.0	0.0	587	-0.4
`19ks`	447.7	1201.8	60.6	1,203	2.0	975.9	0.0	976	18.8
`primary1`	316.5	288.3	16.6	289	6.6	264.0	0.0	264	8.4
`bm1`	309.6	280.4	20.0	284	5.8	233.0	0.0	233	16.9
`primary2`	376.0	1332.2	89.4	1,326	5.4	752.0	0.0	752	43.6
`struct`	165.7	371.4	17.9	370	6.4	346.0	0.0	346	6.8
geo-rand-0.10	514.0	3094.0	0.0	3,094	27.1	3094.0	0.0	3,094	0.0
geo-rand-0.08	533.2	2041.0	0.0	2,041	27.6	2041.0	0.0	2,041	0.0
geo-rand-0.06	433.5	1274.0	0.0	1,274	26.8	1274.0	0.0	1,274	0.0
dsj-geo-rand-0.04	355.5	737.0	0.0	737	24.0	737.0	0.0	737	0.0
dsj-geo-rand-0.02	253.8	222.0	0.0	222	14.9	234.0	0.0	234	-5.4
dsj-geo-rand-0.01	184.1	57.7	6.4	56	4.3	60.0	0.0	60	-4.0
unif-rand-0.10	246.2	21316.6	18.3	21,318	21.6	21342.0	0.0	21,342	-0.1
unif-rand-0.08	240.8	16834.6	16.1	16,835	19.3	16849.0	0.0	16,849	-0.1
unif-rand-0.06	240.2	12182.3	13.7	12,180	20.2	12187.0	0.0	12,187	-0.0
unif-rand-0.04	230.4	7623.4	13.4	7,625	19.0	7652.0	0.0	7,652	-0.4
dsj-unif-rand-0.02	204.4	3412.8	8.5	3,411	17.5	3443.0	0.0	3,443	-0.9
dsj-unif-rand-0.01	171.4	1389.8	4.2	1,390	11.8	1393.0	0.0	1,393	-0.2

Table 11: The performance of Kernighan-Lin and spectral+KL when both are given 55% of our usual computation time.

graphs, although not by as much as time-equated PHC/SG+KL.

From these results, we can safely conclude that KL is the better algorithm of the two, further bolstering its reputation as the standard benchmark.

A.2 Approximate Comparisons

Since we used the standard Johnson et al. [18] benchmark graphs as a subset of our test suite, we can make approximate comparisons against some published algorithms without reimplementing them. There are well-known hazards with this approach, since machine characteristics not captured in a single time-normalization metric may influence algorithm performance, and the degree of code optimization and algorithm parameter tuning cannot be compared. But useful insights can be obtained nonetheless.

Johnson et al. [18] tested simulated annealing (SA) on a DEC VAX 11/750. Our DEC AlphaStation 500/500 is rated at 15.0 on the SPECint95 benchmark. To convert the timings of Johnson et al. and others, we used the following logic: Bui and Moon [5] report that a Sun SparcStation IPX is about 30 times faster than a VAX 11/750, based on Dhrystone tests. The Sun IPX is rated as 21.8 on the SPECint92 benchmark. A recent machine that is rated on both the SPECint92 and SPECint95 benchmarks is the DEC AlphaServer 2100 5/250 used by Battiti and Bertossi [3], at 277.1 and 5.92, respectively. The scores on the two benchmarks differ by a factor of 46.8. Therefore, to compare with our DEC 500/500, the Sun IPX timings should be divided by 15.0/(21.8/46.8) which is 32.2. This means the VAX 11/750 timings should be divided by 32.2×30 which is 966. More easily, we have that the DEC 5/250 timings should be divided by 15.0/5.92 which is 2.53.

Table 12 presents an approximate comparison to the results of Johnson et al. [18, tables VI and XI for geometric graphs and tables IV, III, and II for uniform graphs], giving the following data:

1. *Time*, the originally reported running time on a DEC VAX 11/750.
2. $Time_n$, an estimate of the running time on a DEC AlphaStation 500/500, using the conversion constant derived above.
3. *SA*, the mean solution cost from a single run of simulated annealing as reported by Johnson et al.
4. *KL*, the performance of our KL implementation when given Time_n.

Graph	Time	Time_n	SA	KL	PHC/SG+KL	RGT/SG+KL
dsj-geo-rand-0.04	1038.2	1.1	963.8	739.8	747.4i	**737.0**
dsj-geo-rand-0.02	548.7	0.57	355.3	240.2	251.3i	**227.8**
dsj-geo-rand-0.01	563.7	0.58	120.4	89.1	45.9i	**42.3**
dsj-geo-rand-0.005	539.3	0.56	41.2	45.1	**1.5i**	2.4
dsj-unif-rand-0.02	853.7	0.88	**3402.6**	3439.9	3500.7i	3432.3
dsj-unif-rand-0.01	734.5	0.76	**1376.6**	1403.6	1404.1i	1397.1
dsj-unif-rand-0.005	661.2	0.68	**460.0**	477.9	473.1i	473.7
dsj-unif-rand-0.0025	729.9	0.76	**109.5**	111.9	110.0	110.5

Table 12: Approximate comparison with Johnson et al. [18]. 'i' indicates results taken during population initialization.

5. *PHC/SG+KL*, the performance of PHC/SG+KL when given Time_n. An 'i' after a figure means the algorithm was still initializing its population of 100 solutions when the allotted time expired. The solution value reported by such a run is the best current member of the population. Since KL is only run on selected solutions when the population is fully initialized, such results should probably be regarded as representing the performance of a RGT/SG algorithm.

6. *RGT/SG+KL*, the performance of RGT/SG+KL when given Time_n.

The best figures for each graph, where obvious, are in boldface. These running times range from 2%–20% of the times used in our previous comparisons, which were listed in Table 2. Furthermore, they were determined by observing the 'freezing' of the simulated annealing runs and thus represent a favorable point for annealing in trade-off between time and solution quality. Our results are similar to those of Johnson et al.: annealing excels at random graphs, particularly dense ones, and is surpassed by other algorithms on geometric graphs. The running times are too short for PHC/SG+KL to start its search and its results mimic the longer tests with RGT/SG from Table 4, but its cousin, RGT/SG+KL, consistently beats KL, just as it did in the longer tests shown previously in Table 5.

Johnson et al. also compared the best of 5 runs of annealing to KL [18, tables VI, VII, and XI for geometric graphs and tables IV, V, II for uniform graphs]. We compare against those results in Table 13. KL_J indicates results from Johnson et al.'s KL implementation. These do not match our time-equated KL results for two reasons. First, their implementation

Graph	Time$_n$	5 SAs	KL$_J$	KL	KL$_{I_J}$	PHC/SG+KL	RGT/SG+KL
dsj-geo-rand-0.04	5.4	790.1	737.0	737.0	737.0	737.0	737.0
dsj-geo-rand-0.02	2.8	256.9	224.9	228.3	222.9	225.6	**222.3**
dsj-geo-rand-0.01	2.9	92.5	56.3	66.9	56.5	**39.8**	39.9
dsj-geo-rand-0.005	2.8	35.85	30.3	35.6	NA	**1.0**	1.1
dsj-unif-rand-0.02	4.4	**3392.4**	3480.5	3423.1	3417.6	3409.7	3417.3
dsj-unif-rand-0.01	3.8	**1369.7**	1432.6	1395.0	1388.5	1386.7	1389.6
dsj-unif-rand-0.005	3.4	**454.6**	499.7	471.3	NA	468.2	468.9
dsj-unif-rand-0.0025	3.8	**105.3**	125.0	108.3	NA	107.0	107.4

Table 13: Approximate comparison with Johnson et al. for longer runs.

Graph	Time	Time$_n$	BFS-GBA2.0	KL	PHC/SG+KL	RGT/SG+KL
dsj-geo-rand-0.04	36.99	1.1	738.10	739.8	747.4i	**737.0**
dsj-geo-rand-0.02	32.97	1.0	231.62	236.0	244.3i	**225.9**
dsj-geo-rand-0.01	30.89	0.96	55.78	81.0	40.4	**40.3**
dsj-geo-rand-0.005	17.58	0.55	1.78	45.2	**1.52i**	2.68
dsj-unif-rand-0.02	62.25	1.9	**3401.74**	3427.0	3416.0i	3424.9
dsj-unif-rand-0.01	37.05	1.2	**1376.37**	1400.8	1393.4i	1396.2
dsj-unif-rand-0.005	23.65	0.73	**458.55**	477.3	472.7i	473.7
dsj-unif-rand-0.0025	16.83	0.52	**103.61**	112.6	110.9i	111.5

Table 14: Approximate comparison with Bui and Moon [5]. 'i' indicates results taken during population initialization.

of KL completed 2–4 times more runs during 'Time' then our implementation completed within 'Time$_n$' (the difference seemed to increase with graph sparsity). This could be due to their exclusion of data structure initialization from measurements of running times, subtle algorithmic improvements not reported in their paper, or a slow implementation on our part. KL$_{I_J}$ indicates the results of our implementation when given the same number of runs as Johnson et al.'s. Although we then obtained similar performance on the geometric graphs, we report much better results on the uniform random graphs. This may be due to our programming of the buckets data structure. Hagen, Huang, and Khang [14] have shown that a LIFO insertion order can significantly improve solution quality.

With the longer running time, PHC/SG+KL begins to pull away from KL. Annealing maintains its advantage on the uniform graphs, however.

Graph	Time	Time_n	RRTS	KL	PHC/SG+KL	RGT/SG+KL
dsj-geo-rand-0.04	2.02	0.80	737.0	742.4	751.5i	737.2
dsj-geo-rand-0.02	1.01	0.40	**222.27**	248.1	252.0i	228.6
dsj-geo-rand-0.01	0.59	0.23	**39.76**	105.6	49.4i	47.8
dsj-geo-rand-0.005	0.25	0.10	**1.0**	58.1	3.6i	4.9
dsj-unif-rand-0.02	1.10	0.43	**3389.64**	3448.9	3509.2i	3437.2
dsj-unif-rand-0.01	0.67	0.26	**1371.47**	1413.3	1450.1i	1403.6
dsj-unif-rand-0.005	0.47	0.19	**458.32**	484.5	501.4i	480.9
dsj-unif-rand-0.0025	0.49	0.19	**107.18**	116.1	122.1i	114.5

Table 15: Approximate comparison with Battiti and Bertossi's RRTS algorithm [3], given a short amount of time.

Graph	Time	Time_n	RRTS	KL	PHC/SG+KL	RGT/SG+KL
dsj-geo-rand-0.04	24.3	9.6	737.0	737.0	737.0	737.0
dsj-geo-rand-0.02	12.5	4.9	222.0	224.5	224.0	222.0
dsj-geo-rand-0.01	6.3	2.5	**39.03**	68.4	39.8	40.2
dsj-geo-rand-0.005	4.2	1.7	1.0	37.5	1.0	1.4
dsj-unif-rand-0.02	14.7	5.8	**3383.92**	3421.3	3408.9	3413.2
dsj-unif-rand-0.01	9.3	3.7	**1364.27**	1395.0	1387.2	1389.9
dsj-unif-rand-0.005	6.5	2.6	**450.99**	471.8	468.8	470.3
dsj-unif-rand-0.0025	6.5	2.6	**98.69**	108.7	107.4	108.0

Table 16: Approximate comparison with RRTS for longer running times.

Bui and Moon have also published empirical results on the Johnson et al. graphs [5, Table II]. Their algorithm, called BFS-GBA2.0 (for 'genetic bisection algorithm' with 'best first search' preprocessing), combines a specialized genetic algorithm with a truncated version of KL. Table 14 shows a time-equated comparison with their results, which were originally timed on a Sun SparcStation IPX. BFS-GBA2.0 surpasses KL on all graphs. It is beaten by RGT/SG+KL on most geometric graphs, however. PHC/SG+KL's selective application of KL seems to give an advantage on the sparsest geometric graph. BFS-GBA2.0 dominates the seed-growth algorithms on the uniform graphs. Interestingly, it does only slightly better than Johnson et al.'s SA on most of them, even though it has been given more time.

In Tables 15 and 16, we present a comparison to Battiti and Bertossi's RRTS algorithm (from 'reactive randomized tabu search'), which is based on

a constructive heuristic similar to seed-growth in which the number of edges to the partition being added to is used only to break ties [3, Table 5]. They combine the heuristic, which they call Min-Max-Greedy, with a long tabu-search refinement phase. Their timings were originally measured on a DEC AlphaServer 2100 5/250, and do not include data-structure initialization times. Table 15 shows the performance of the algorithms for short running times. RRTS seems to dominate all other algorithms we have considered, although implementation tuning may play a significant role at these short running times. When given more time, seed-growth with KL refinement gives similar performance on geometric graphs, although the tabu refinement seems to give RRTS a decided advantage on the uniform graphs. Intuitively, this fits into the pattern noted by Johnson et al., because relative to KL, tabu search shares much in common with simulated annealing, which performs very well on random graphs. In later work, Battiti and Bertossi [2] show that RGT/SG (which they call Diff-Greedy) is superior to RGT/Min-Max-Greedy (RRTS without its tabu postprocess), although they do not discuss adding refinement techniques.

These comparisons, although approximate, have shown that many recently proposed algorithms have surpassed time-equated KL, and that methods based on seed-growth may be the most effective. The advantage of seed-growth methods seems to vary with the refinement technique used. Both KL and tabu-search refinement work well on the geometric graphs, but tabu search does much better on random graphs. It is clear that these hybrid approaches that combine two style of optimization are superior to any single technique used alone, even when compared fairly.

References

[1] E. R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal of Algebraic and Discrete Methods*, 3(4):541–550, 1982.

[2] R. Battiti and A. Bertossi. Differential greedy for the 0-1 equicut problem. In D. Z. Du and P. M. Pardalos, editors, *Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilities Location*, 1997.

[3] R. Battiti and A. Bertossi. Greedy and prohibition-based heuristics for graph partitioning. Technical Report UTM-97-512, Università di Trento, Dipartimento di Matematica, Trento, Italy, February 1997. Available via WWW.

[4] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 775–778, 1989.

[5] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45(7):841–855, 1996.

[6] J. Cong and M. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 755–760, Dallas, TX, June 1993.

[7] M. Dell'Amico and F. Maffioli. A new tabu search approach to the 0-1 equicut problem. In *Metaheuristics 1995: The State of the Art*, pages 361–377. Kluwers Academic Publishers, 1996.

[8] W. E. Donath. Logic partitioning. In B. Preas and M. Lorenzetti, editors, *Physical Design Automation of VLSI Systems*, pages 65–86. Benjamin/Cummings, 1988.

[9] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitioning. In *Proceedings of the 19th Design Automation Conference*, pages 175–181, Las Vegas, NM, 1982.

[10] J. Frankle and R. M. Karp. Circuit placement and cost bounds by eigenvector decomposition. In *Proceedings of the IEEE International COnference on Computer-Aided Design*, pages 414–417, 1986.

[11] J. Garbers, H. J. Prömel, and A. Steger. Finding clusters in VLSI circuits. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 520–523, Santa Clara, California, Nov. 1990.

[12] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[13] M. K. Goldberg and M. Burstein. Heuristic improvement technique for bisection of VLSI networks. In *Proceedings of the IEEE International Conference on Computer Design*, pages 122–125, Port Chester, NY, 1983.

[14] L. Hagen, D. J.-H. Huang, and A. B. Khang. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on Computer-Aided Design*, in press.

[15] L. Hagen and A. B. Kahng. Fast spectral methods for ratio cut partitioning and clustering. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 10–13, 1991.

[16] L. Hagen and A. B. Kahng. A new approach to effective circuit clustering. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 422–427, Santa Clara, California, Nov. 1992.

[17] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 13(2):311–329, 1970.

[18] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; Part I, graph partitioning. *Operations Research*, 37(6):865–892, November-December 1989.

[19] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

[20] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984.

[21] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[22] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, C-33:438–446, 1984.

[23] M. Laguna, T. A. Feo, and H. C. Elrod. A greedy randomized adaptive search procedure for the two-partition problem. *Operations Research*, 42(4):677–687, July-August 1994.

[24] J. Marks, W. Ruml, S. M. Shieber, and J. T. Ngo. A seed-growth heuristic for graph bisection. In R. Battiti and A. A. Bertossi, editors, *Proceedings of the Workshop on Algorithms and Experiments (ALEX98)*, February 1998.

[25] J. Marks, S. M. Shieber, and J. T. Ngo. A stochastic search technique for graph bisection. Technical Report TR94-18, MERL—A Mitsubishi Electric Research Laboratory, Cambridge, MA, November 1994.

[26] T.-K. Ng, J. Oldfield, and V. Pitchumani. Improvements of a mincut partition algorithm. In *Proceedings of the IEEE International Conference on Computer Design*, pages 470–473, Santa Clara, CA, 1987.

[27] Y.-C. Wei and C.-K. Cheng. A two-level two-way partitioning algorithm. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 516–519, Santa Clara, CA, Nov. 1990.

[28] Y.-C. Wei and C.-K. Cheng. Ratio cut partitioning for hierarchical design. *IEEE Transactions on Computer-Aided Design*, 10(7):911–921, July 1991.