Proceedings of "Algorithms and Experiments" (ALEX98) Trento, Italy, Feb 9–11, 1998
R. Battiti and A. A. Bertossi (Eds.) pp. 76-87

A Seed-Growth Heuristic for Graph Bisection

Joe Marks
MERL–A Mitsubishi Electric Research Laboratory
Cambridge, MA 02139, USA
e-mail: marks@merl.com

Wheeler Ruml
Division of Engineering and Applied Sciences, Harvard University
Cambridge, MA 02138, USA
e-mail: ruml@eecs.harvard.edu

Stuart M. Shieber
Division of Engineering and Applied Sciences, Harvard University
Cambridge, MA 02138, USA
e-mail: shieber@eecs.harvard.edu

and

J. Thomas Ngo
Interval Research Corporation
Palo Alto, CA 94304-1216, USA
e-mail: ngo@interval.com

Abstract

We present a new heuristic algorithm for graph bisection, based on an implicit notion of clustering. We describe 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 with large-sample runs of pure Kernighan-Lin, the new algorithm demonstrates significant superiority in terms of the best bisections found.

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 [7]. Graph bisection and its generalizations[1] have 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 [13]. (The efficient implementation of this heuristic technique was described by Fiduccia and Mattheyses [5], 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 subsets X and Y. The vertex-swap operation is also the primitive perturbation operator used in applications of simulated annealing to graph bisection [14, 15].

[1] More general classes of graph-partitioning problems arise when V can be divided into more than two subsets, when the strict equality constraint on the sizes of the subsets is relaxed, and when weights are associated with the vertices and edges to be used in the constraint-satisfaction and cut-set-size computations.

In spite of the folk wisdom that simulated annealing is capable of avoiding the local minima that often plague greedy heuristics like the KL algorithm, Johnson et al. [12] found that the relative performance of the two algorithms depends on the nature of the graphs being bisected: simulated annealing has an advantage on sparse, relatively uniform graphs, but KL is better for graphs with structure.[2]

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 [2, 3, 4, 6, 9, 11, 16, 17, 18], 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 algorithm we describe in this paper can be considered a synthesis of ideas from previous work: it includes a very simple implicit clustering heuristic, employs a stochastic search strategy (like simulated annealing or a genetic algorithm [8]), and uses 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 algorithm exhibits significant superiority on a variety of test graphs.

In the following sections we describe the algorithm, present an empirical analysis of its behavior, and conclude with a discussion of future work.

2 Algorithm Description

Our algorithm is based on a simple *seed-growth* heuristic.[3] 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 $\mathrm{cut}(\{a\}, Y) - \mathrm{cut}(\{a\}, X)$; 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 notion of clustering is implicit in this heuristic, as compared to heuristics in which explicit clusters are computed and manipulated [2, 3, 4, 6, 9, 11, 16, 17, 18].

More formally, the algorithm can be given by the following pseudocode. (All underlined quantities are parameters of the heuristic that can be varied. The values given in the paper are those that gave the best empirical results in an initial set of experiments.)

Input: An undirected graph $G = (V, E)$. $|V|$ is assumed to be even.

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 $|s_x| = |s_y| = \lfloor \underline{0.01}\,|V| \rfloor$.
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 $\mathrm{cut}(\{a\}, Y) - \mathrm{cut}(\{a\}, X)$ is minimal. $X \leftarrow X \cup \{a\}$.
   (b) Find an unassigned vertex $b \in V$ such that $\mathrm{cut}(\{b\}, X) - \mathrm{cut}(\{b\}, Y)$ is minimal. $Y \leftarrow Y \cup \{b\}$.

[2] The conclusions that Johnson and his colleagues drew from their thorough empirical analysis are more complicated and informative than this simple précis suggests, but the statement is approximately true.

[3] This heuristic bears some resemblance to the epitaxial-growth heuristic of Donath [4].

One application of the seed-growth heuristic is not likely to be particularly useful (on average it will be worse than a single application of the KL algorithm), but the $O(|V| + |E|)$ seed-growth heuristic—which is roughly five times faster than an efficient implementation of the KL algorithm on standard test graphs—can be rendered effective by running it many times as part of a general search procedure. One such search procedure, a form of parallel hill climbing, is given here, though others (e.g., simulated annealing and genetic algorithms) might also be used effectively in combination with the seed-growth heuristic. The KL algorithm can be used as a postprocess to achieve final refinement of promising bisections found by the search procedure.

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) $\underline{2,500}$ times (or until the alloted 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 seed-growth (SG) heuristic, and the KL algorithm, we will refer to it as PHC/SG+KL.

3 Empirical Analysis

Heuristic algorithms for graph partitioning like the one described here 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 our algorithm.

	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.

3.1 Caveats

Consider the evidence presented in Table 1. (This example is based on an empirical analysis reported by Wei and Cheng [18].) The table contains the average and minimum cut-set sizes of 9 graph bisections, computed from 20 runs of the KL algorithm and 20 runs of Algorithm X.[4] 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 circuit in the standard UCLA benchmark suite. Suppose that Algorithm Y also 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. By inspection, roughly 1% of the values in the histogram for KL are less than 300. A simple probabilistic analysis shows that n must be around 690 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 wait the hour or so required for 1,000 runs of KL—as is typical for many applications involving graph partitioning—KL should be considered the better algorithm on the basis of this empirical evidence: it will very likely find a bisection with a smaller cut set than Algorithm Y. 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 [17]. Conversely, any comparisons with KL that involve 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 [3, 11, 18, 19].

3.2 Results

[4] The graphs were derived from circuit hypergraphs widely used as benchmarks in the VLSI CAD community. They are available at `http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html`.

	Graph				KL			PHC/SG+KL		
Name	$\|V\|$	$\overline{\deg}$	$p(edge)$	Time	Runs	Mean	σ	Mean	σ	Impr.
test4	1,515	166.8	0.11017	212.8	553.0	1263.1	11.4	1245.5	9.2	1.4
test5	2,595	211.3	0.08146	499.3	566.0	2032.2	33.5	1953.1	9.3	3.9
test6	1,752	139.8	0.07983	228.5	639.8	1207.8	9.5	1188.4	3.5	1.6
fract	149	11.7	0.07881	2.5	481.4	55.0	0.0	55.0	0.0	0.0
test2	1,663	126.8	0.07630	206.0	603.4	1242.8	11.6	1243.2	17.5	-0.0
test3	1,607	81.4	0.05071	114.1	586.4	911.3	17.1	828.2	1.9	9.1
balu	801	38.6	0.04828	31.5	462.4	584.8	0.8	584.1	0.3	0.1
19ks	2,844	130.8	0.04600	317.7	568.7	1177.7	58.1	985.7	37.3	16.3
primary1	833	15.0	0.01806	15.1	430.6	281.4	17.3	218.0	1.4	22.5
bm1	882	14.2	0.01611	15.3	418.5	275.0	19.0	212.9	4.1	22.6
primary2	3,014	24.7	0.00820	79.6	490.8	1322.9	81.7	585.4	27.0	55.7
struct	1,952	8.8	0.00449	22.4	226.5	367.2	16.6	331.8	7.5	9.6
industry3	15,406	23.3	0.00152	408.7	295.6	6827.2	294.8	990.0	132.2	85.5
s9234	5,866	5.5	0.00093	53.9	201.5	667.1	25.6	189.1	18.9	71.7
s13207	8,772	6.5	0.00074	93.6	203.2	803.4	40.2	201.3	26.9	74.9
s38584	20,995	13.7	0.00065	383.8	240.5	3518.4	171.3	554.3	72.8	84.2
s15850	10,470	6.3	0.00060	107.5	181.1	985.9	43.5	252.7	34.1	74.4
s38417	23,949	7.2	0.00030	291.7	195.8	2280.4	73.0	546.6	71.6	76.0
geo-0.01	1,000	9.4	0.00942	13.3	232.3	45.8	6.5	39.0	0.0	14.8
geo-0.02	1,000	18.6	0.01858	20.9	280.4	186.0	0.0	186.0	0.0	0.0
geo-0.04	1,000	36.4	0.03643	37.0	388.9	583.0	0.0	583.0	0.0	0.0
geo-0.06	1,000	53.2	0.05321	53.5	452.5	1274.0	0.0	1274.0	0.0	0.0
geo-0.08	1,000	72.3	0.07234	71.6	551.1	2041.0	0.0	2041.0	0.0	0.0
geo-0.10	1,000	85.3	0.08535	80.6	509.9	3094.0	0.0	3094.0	0.0	0.0
unif-0.01	1,000	10.0	0.01001	13.9	227.1	1359.8	5.2	1351.6	5.2	0.6
unif-0.02	1,000	20.2	0.02021	23.1	239.6	3423.6	9.1	3410.9	8.0	0.4
unif-0.04	1,000	39.6	0.03968	41.3	250.5	7622.8	10.2	7608.4	9.6	0.2
unif-0.06	1,000	59.7	0.05979	62.2	265.2	12180.5	13.3	12166.8	10.5	0.1
unif-0.08	1,000	80.1	0.08016	79.3	253.3	16838.9	17.2	16814.7	14.4	0.1
unif-0.10	1,000	99.2	0.09933	101.0	266.6	21309.2	14.6	21301.6	10.4	0.0

Table 2: Kernighan-Lin and PHC/SG+KL: an empirical comparison.

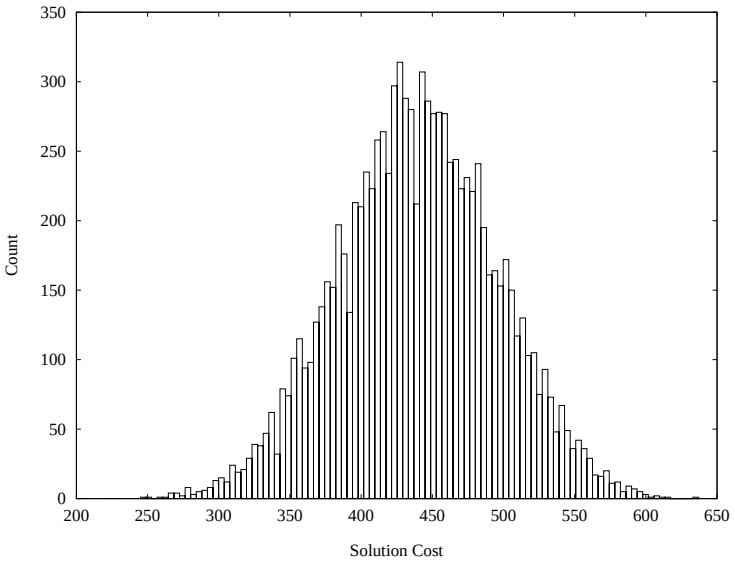

Figure 1: Histogram of solutions computed by the KL algorithm for graph `bm1`.

Table 2 contains an empirical comparison of the KL and PHC/SG+KL. The algorithms were tested on three classes of problems: 18 graphs derived from VLSI benchmark circuit hypergraphs, 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 [12]. The names for the random graphs indicate the expected probability of existence of a possible edge.[5] The values we use were chosen to match the range of edge probabilities in the circuit graphs, which we take as representative of important practical problems.

For each graph in our test suite, the following data are presented:

1. *Graph cardinality:* The number of vertices in the graph ($|V|$).
2. *Mean degree:* The average number of edges incident upon a vertex in the graph.
3. *Edge probability:* The probability of a possible edge appearing in the graph.
4. *Running time:* The running time allowed for each algorithm on the graph, in seconds on a DEC AlphaStation 500/500. This was computed by timing 2,500 iterations of the PHC/SG algorithm (as explained below, this variant is just PHC/SG+KL without the KL refinement steps). The running times range from 2.5 seconds for graph `fract` to 8.3 minutes for graph `test5`.
5. *Number of KL runs:* The number of runs of the KL algorithm that will take an amount of time equivalent to that required for the PHC/SG algorithm.
6. *Average 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 required for time equivalence with the PHC/SG algorithm.
7. *Standard deviation of minimum cut-set size for KL:* The standard deviation of the minimum cut-set size found over the 25 tests.

[5] As Johnson et al. explain, for a geometric random graph with expected degree d, one uses a radius of $\sqrt{d/(|V|\pi)}$.

8. *Average minimum cut-set size for PHC/SG+KL:* The average minimum cut-set size found over 25 runs of the PHC/SG+KL algorithm.

9. *Standard deviation of minimum cut-set size for PHC/SG+KL:* The standard deviation of the minimum cut-set sizes found over the 25 tests.

10. *Improvement over KL:* The average improvement of the PHC/SG+KL algorithm over the KL algorithm, expressed as a percentage of the average minimum cut-set size for KL.

In all cases, PHC/SG+KL generates solutions that are at least as good as those from the large-sample, time-equated tests of KL. The reduction in the size of the cut set ranges from none to 85%.

The results for PHC/SG+KL may appear ordinary relative to the results that have been reported recently for various clustering heuristics.[6] However, this is due in large part to the better results we report for KL because of the large number of KL runs we use, over 300 on average. Recall that 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 300 runs is quite an improvement, on average, over the best of 100 runs. Thus, our results cannot be directly compared to those previously published. Preliminary experiments with an implementation of one of the so-called spectral methods for graph-bisection [1, 10] indicates that it fares worse than time-equated KL. We hope to complete a full comparison with other algorithms in the near future.

An interesting aspect of the data is the variation in relative performance of the algorithms: although PHC/SG+KL is superior to KL across the board, the degree of superiority differs markedly. For some graphs (`balu`, `Test02`, `Test04`, and `Test06`) the improvement is very small, yet for others (`primary2`, `s28584`, and `industry3`, for example) the improvement is substantial. We will investigate these differences further below.

For hybrid algorithms that involve the KL algorithm as a postprocess, the following question naturally arises: How much work is the KL part doing? Table 3 presents results of the PHC/SG algorithm, which is the same as PHC/SG+KL, but without the KL postprocess refinement in steps 3 and 4d. PHC/SG still returns substantially better results than KL for many graphs, but it does not exhibit the consistent superiority of PHC/SG+KL. One possible explanation could be that 2,500 iterations is simply not long enough for the search procedure to discover effective seed sets. We can examine the progress of the search to test this hypothesis.

Figure 2 shows a trace of the progress of the three algorithms on graph `test3`. Error bars represent standard deviations over 25 runs, calculated separately above and below the mean. It is clear from the plot (which is typical of those graphs for which PHC/SG is not markedly superior to KL) that the search procedure is still making progress when it is cut short after 2,500 iterations. Furthermore, this close analysis reveals that the KL refinement is effective even without the search procedure (although it does improve detectably with additional time). Although our previous comparison, allowing time for hundreds of runs of KL, more closely reflects applications in which quality is paramount, there are situations in which computation time is the limiting resource. Figure 3 summarizes the performance of the three algorithms on the circuit-derived graphs when given just enough time for PHC/SG+KL to initialize its population (about 4% of the times listed in table 2). Each result is plotted according to the graph's edge probability. Solution costs have been normalized against KL, so -10 refers to a 10% reduction in the size of the cut set. The plot highlights a significant feature of our algorithm: there seems to be a correlation between the edge probability in the graph and the improvement it exhibits over the Kernighan-Lin algorithm. Figures 4–6 summarize the information presented previously in tables 2 and 3 and confirm that the edge probability correlation holds when the algorithms are given abundant computation time, and for all three classes of graphs examined.

[6] Unfortunately a direct comparison with other algorithms on the circuit graphs based on published figures is not currently possible, because the common convention is to report cut-set size in terms of nets (edges in a hypergraph) rather than edges in the graph derived from the original hypergraph, which is what we have done here for consistency with other presentations [2, 9, 12]. Furthermore, we bisect the graph on the basis of the number of vertices in each half of the bisection, not the weighted sum of the areas associated with them.

	Graph				KL			PHC/SG		
Name	$\|V\|$	$\overline{\deg}$	$p(edge)$	Time	Runs	Mean	σ	Mean	σ	Impr.
`test4`	1,515	166.8	0.11017	212.8	553.0	1263.1	11.4	1267.9	19.6	-0.4
`test5`	2,595	211.3	0.08146	499.3	566.0	2032.2	33.5	2116.8	75.5	-4.2
`test6`	1,752	139.8	0.07983	228.5	639.8	1207.8	9.5	1220.7	15.1	-1.1
`fract`	149	11.7	0.07881	2.5	481.4	55.0	0.0	55.0	0.0	0.0
`test2`	1,663	126.8	0.07630	206.0	603.4	1242.8	11.6	1260.4	18.7	-1.4
`test3`	1,607	81.4	0.05071	114.1	586.4	911.3	17.1	862.5	21.8	5.4
`balu`	801	38.6	0.04828	31.5	462.4	584.8	0.8	585.9	0.6	-0.2
`19ks`	2,844	130.8	0.04600	317.7	568.7	1177.7	58.1	1203.2	158.6	-2.2
`primary1`	833	15.0	0.01806	15.1	430.6	281.4	17.3	224.8	4.6	20.1
`bm1`	882	14.2	0.01611	15.3	418.5	275.0	19.0	219.2	4.9	20.3
`primary2`	3,014	24.7	0.00820	79.6	490.8	1322.9	81.7	745.4	41.0	43.7
`struct`	1,952	8.8	0.00449	22.4	226.5	367.2	16.6	347.7	10.7	5.3
`industry3`	15,406	23.3	0.00152	408.7	295.6	6827.2	294.8	2788.6	198.1	59.2
`s9234`	5,866	5.5	0.00093	53.9	201.5	667.1	25.6	287.3	33.3	56.9
`s13207`	8,772	6.5	0.00074	93.6	203.2	803.4	40.2	335.6	31.8	58.2
`s38584`	20,995	13.7	0.00065	383.8	240.5	3518.4	171.3	1935.5	155.3	45.0
`s15850`	10,470	6.3	0.00060	107.5	181.1	985.9	43.5	471.7	34.7	52.2
`s38417`	23,949	7.2	0.00030	291.7	195.8	2280.4	73.0	1377.5	94.8	39.6
geo-0.01	1,000	9.4	0.00942	13.3	232.3	45.8	6.5	40.2	1.4	12.3
geo-0.02	1,000	18.6	0.01858	20.9	280.4	186.0	0.0	186.0	0.0	0.0
geo-0.04	1,000	36.4	0.03643	37.0	388.9	583.0	0.0	585.4	3.1	-0.4
geo-0.06	1,000	53.2	0.05321	53.5	452.5	1274.0	0.0	1276.0	2.3	-0.2
geo-0.08	1,000	72.3	0.07234	71.6	551.1	2041.0	0.0	2041.0	0.0	0.0
geo-0.10	1,000	85.3	0.08535	80.6	509.9	3094.0	0.0	3094.8	2.2	-0.0
unif-0.01	1,000	10.0	0.01001	13.9	227.1	1359.8	5.2	1392.4	5.5	-2.4
unif-0.02	1,000	20.2	0.02021	23.1	239.6	3423.6	9.1	3480.7	8.8	-1.7
unif-0.04	1,000	39.6	0.03968	41.3	250.5	7622.8	10.2	7718.0	9.0	-1.2
unif-0.06	1,000	59.7	0.05979	62.2	265.2	12180.5	13.3	12297.0	17.2	-1.0
unif-0.08	1,000	80.1	0.08016	79.3	253.3	16838.9	17.2	16963.2	15.3	-0.7
unif-0.10	1,000	99.2	0.09933	101.0	266.6	21309.2	14.6	21465.4	16.2	-0.7

Table 3: Kernighan-Lin and PHC/SG.

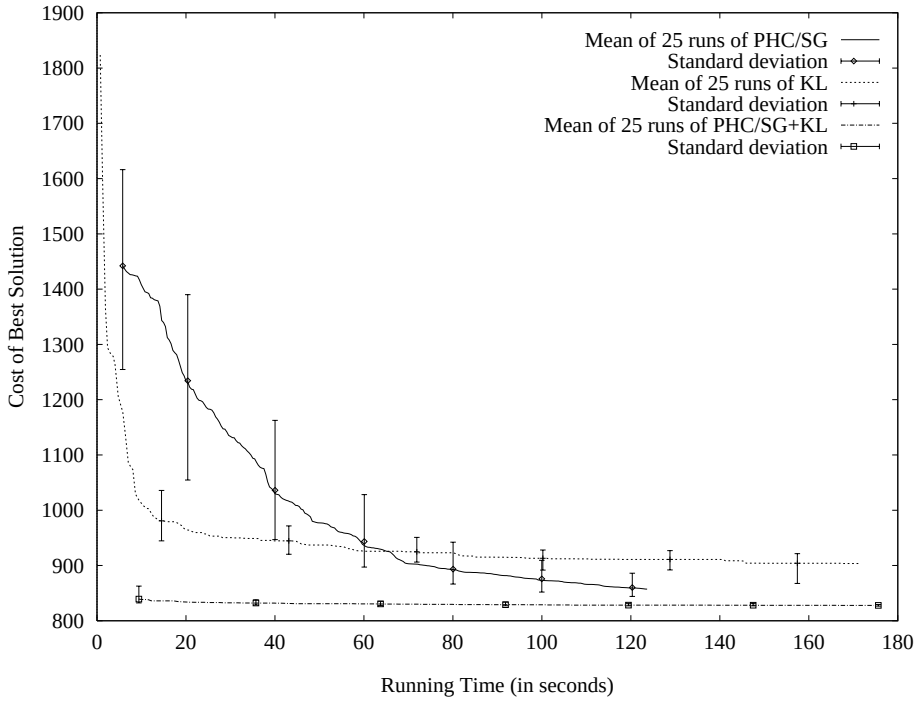

Figure 2: Performance of the three algorithms on graph `test3`.

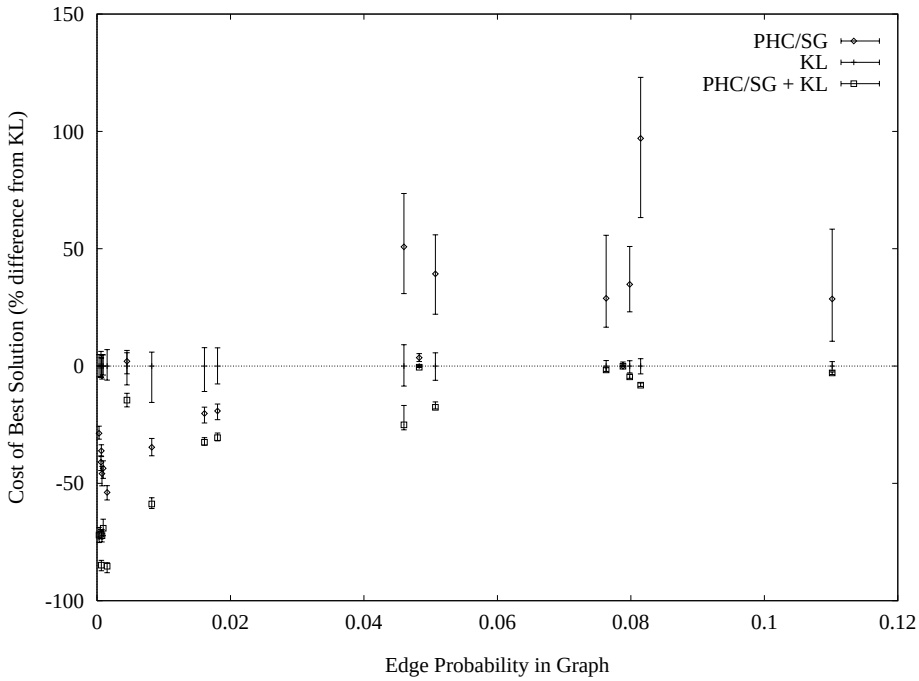

Figure 3: Performance of the three algorithms on the circuit graphs when given very little time.

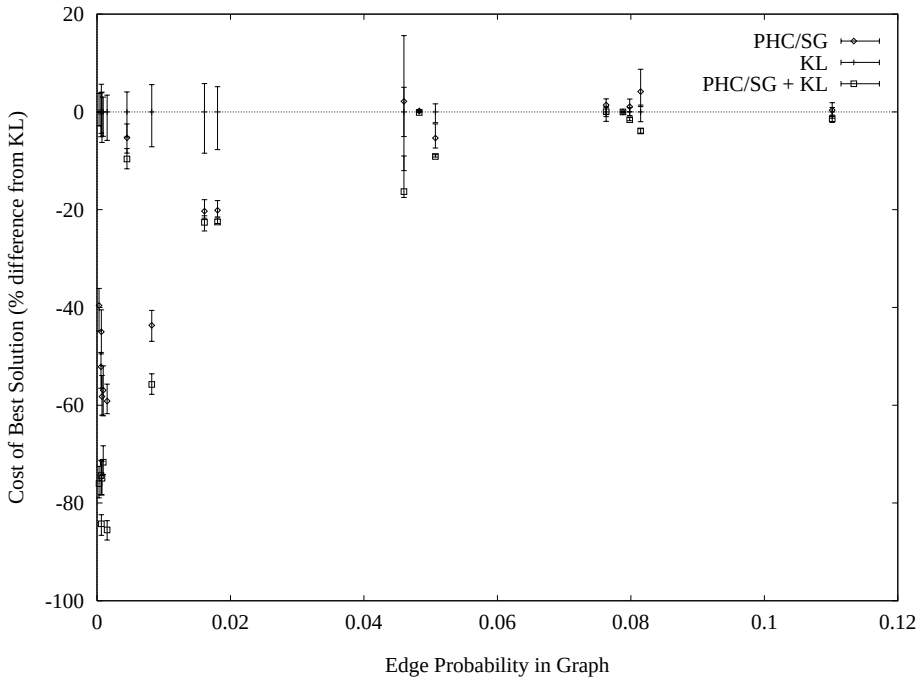

Figure 4: Summary of the three algorithms on the circuit graphs.

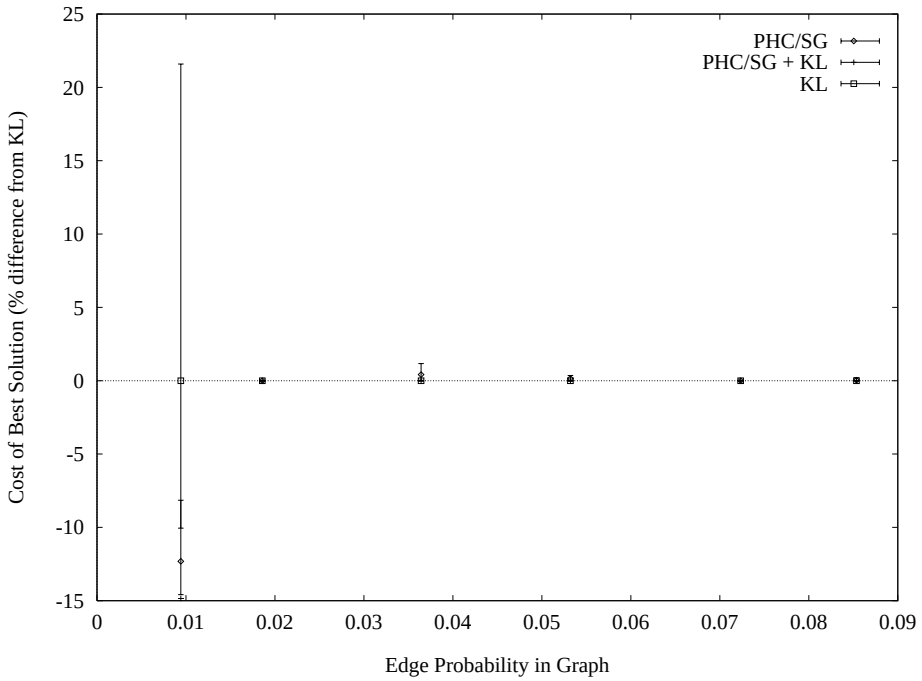

Figure 5: Summary of the three algorithms on geometric random graphs.

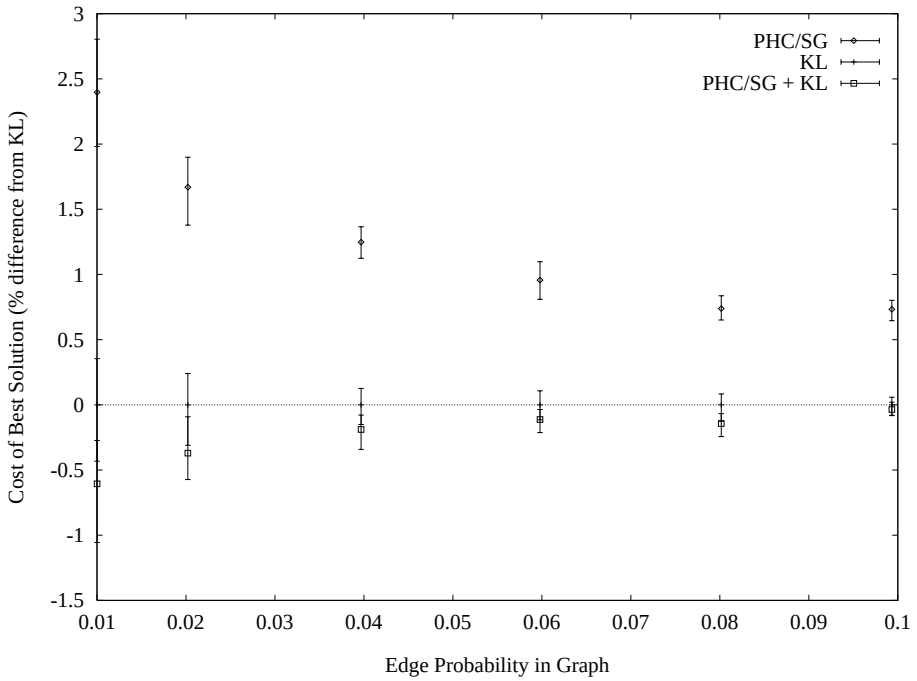

Figure 6: Summary of the three algorithms on uniform random graphs.

4 Conclusions

The PHC/SG+KL algorithm is undoubtedly 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. 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 different algorithms.

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.

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] 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.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, New York, 1989.

[9] 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.

[10] 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.

[11] 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.

[12] 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, Nov.-Dec. 1989.

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

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

[15] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

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

[17] 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.

[18] 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.

[19] 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.