A Stochastic Search Technique for Graph Bisection

Joe Marks, Stuart Shieber, J. Thomas Ngo

TR94-18 December 1994

Abstract

We present a new heuristic algorithm for graph bisection. This heuristic combines stochastic search and an implicit notion of clustering in a novel manner. In comparison with a large-sample, time-equated set of runs of the Kernighan-Lin algorithm, the new algorithm demonstrates a modest but significant superiority in terms of the best bisections found. Keywords and phrases: graph bisection, graph partitioning, stochastic search, heuristics.

1. First printing, TR94-18, November 1994.

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-sized 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 $\text{cut}(X, Y)$) is minimized. This problem is NP-complete [6]. 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 [11]. (The efficient implementation of this heuristic technique was described by Fiduccia and Mattheyses [4], 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 [12, 13]. 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. [10] 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 [1, 2, 3, 5, 8, 9, 14, 15, 16], 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 [7]), 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.

[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.

[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.

2 Algorithm Description

Our algorithm is based on a simple *seed-growth* heuristic.[3] We start with two disjoint, equal-sized 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 $\text{cut}(\{a\}, Y) - \text{cut}(\{a\}, X)$; 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 [1, 2, 3, 5, 8, 9, 14, 15, 16].

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 $\text{cut}(\{a\}, Y) - \text{cut}(\{a\}, X)$ 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 min-
       imal.
       $Y \leftarrow Y \cup \{b\}$.
```

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 an order of magnitude 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 is used as a postprocess to achieve final refinement of the best bisections found by the search procedure.

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

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. Repeat substeps (a) through (e) $\underline{5,000}$ times:
   (a) Randomly pick a seed-set pair $(s_x, s_y) \in P$.
   (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) If $\text{cut}(X', Y') < \text{cut}(X, Y)$ then replace (s_x, s_y) in P with (s'_x, s'_y).
   (e) Every $\underline{1,000}$th iteration perform the following steps:
      i. Use the cut-set sizes of the corresponding bisections (i.e., the values of $\text{cut}(X, Y)$) to rank order the seed-set pairs (s_x, s_y) in P.
      ii. Replace the bottom $\underline{50}$ seed-set pairs in P with copies of the top $\underline{50}$ seed-set pairs in P.
4. Use the cut-set sizes of the corresponding bisections to rank order the seed-set pairs (s_x, s_y) in P.
5. For the top $\underline{20\%}$ of seed-set pairs (s_x, s_y) in P apply the KL algorithm to (X, Y); return the best bisection found.

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
`19ks`	1131	1701.90	1154	1391.40	-2.03	18.24
`5655`	633	866.90	608	698.70	3.95	19.40
`8870`	70	118.15	69	95.10	1.43	19.51
`PrimGA1`	312	384.10	293	345.65	6.09	10.01
`PrimGA2`	1262	1716.30	915	1405.50	27.50	18.11
`Test02`	1177	1296.60	1195	1242.10	-1.53	4.20
`Test03`	906	2590.30	843	1503.60	6.95	41.95
`Test04`	1216	1316.35	1201	1245.55	1.23	5.38
`Test05`	2119	4524.55	1866	2113.95	11.94	53.28
`Test06`	1203	1580.10	1192	1285.95	0.91	18.62
`bm1`	302	385.10	229	327.80	24.17	14.88

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 [16].) The table contains the average and minimum cut-set sizes of 11 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 MCNC test 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 250 and 300 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 250. A simple probabilistic analysis shows that n

[4]The graphs were derived from circuit hypergraphs that were made available for the Microelectronics Center of North Carolina (MCNC) Layout Synthesis Workshop.

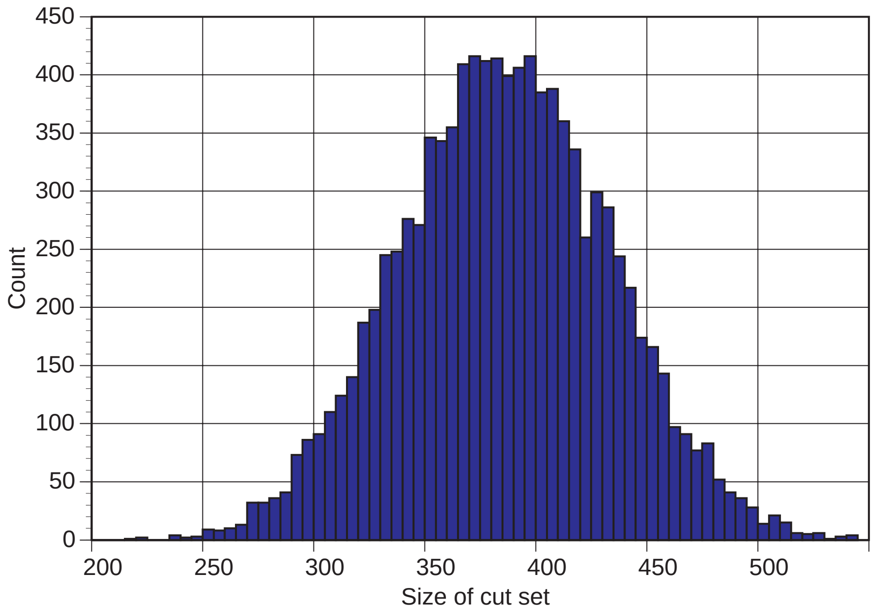

Figure 1: Histogram of values computed by the KL algorithm for graph `bm1`.

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 1000 runs of KL—as is typical for most 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 [15]. 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 [2, 9, 16, 17].

3.2 Results

Table 2 contains an empirical comparison of the KL and PHC/SG+KL algorithms. The algorithms were tested on 13 graphs, 11 of which were derived from hypergraphs in the MCNC test suite, and two of which have been used for empirical testing in the operations-research community.[5]

[5]These graphs are instances of $G(1000, 0.0025)$ and $U(1000, 0.04)$. Graphs in $G(n, p)$ have n vertices, and the probability that there is an edge between any given pair of vertices is p. Graphs in $U(n, d)$ have n vertices that are randomly distributed on a unit square, and an edge exists between any pair of vertices that are distance d or less apart. One would expect the graphs in U, but not the graphs in G, to have exploitable structure [10].

				KL			PHC/SG+KL		
		avg	Time	# of	avg min		avg min		% impr.
Graph	$\|V\|$	deg	(secs)	runs	cut-set size	σ	cut-set size	σ	over KL
`19ks`	2844	93.2	12368	512	1020.8	33.5	976.8	89.2	4.3%
`5655`	922	20.1	1289	702	603.2	4.3	595.4	0.5	1.3%
`8870`	502	9.7	377	728	52.8	1.3	52.0	0.0	1.5%
`PrimGA1`	834	11.3	1054	628	235.6	15.6	218.8	0.8	7.1%
`PrimGA2`	3014	18.0	13785	420	1051.6	77.5	574.6	31.2	45.4%
`Test02`	1664	100.1	4245	486	1172.8	11.9	1164.2	22.1	0.7%
`Test03`	1608	71.2	3981	608	821.8	8.1	804.4	0.5	2.1%
`Test04`	1516	137.1	3589	454	1191.2	2.2	1184.0	3.1	0.6%
`Test05`	2596	167.3	10409	420	1887.4	26.4	1813.0	2.4	3.9%
`Test06`	1752	114.7	4718	500	1194.4	3.6	1188.4	2.3	0.5%
`bm1`	882	10.7	1176	570	240.4	14.5	209.2	1.3	13.0%
$G(1000, 0.0025)$	1000	2.5	1538	234	98.2	3.0	93.8	1.5	4.5%
$U(1000, 0.04)$	1000	5.0	1507	380	28.6	5.7	4.2	0.4	85.3%

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

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

1. *Graph cardinality:* The number of vertices in the graph ($|V|$).
2. *Average degree:* The average number of edges incident upon a vertex in the graph.
3. *Running time:* The running time, in seconds, of the PHC/SG+KL algorithm on a Hewlett-Packard 735 workstation. (The running times range from a little under four hours for graph `PrimGA2` to a little over six minutes for graph `8870`.)
4. *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+KL algorithm.
5. *Average minimum cut-set size for KL:* The average minimum cut-set size found over five tests of k runs each, where k is the number of runs required for time equivalence with the PHC/SG+KL algorithm.
6. *Standard deviation of minimum cut-set size for KL:* The standard deviation of the minimum cut-set size found over the five tests.
7. *Average minimum cut-set size for PHC/SG+KL:* The average minimum cut-set size found over five runs of the PHC/SG+KL algorithm.
8. *Standard deviation of minimum cut-set size for PHC/SG+KL:* The standard deviation of the minimum cut-set size found over the five tests.

9. *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 better solutions than the large-sample, time-equated tests of KL. The advantage ranges from less than 1% to over 85%.

The results for PHC/SG+KL may appear modest 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, on average about 500. 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 500 runs is quite an improvement, on average, on the best of 100 runs. Thus, our results cannot be directly compared to those previously published. We hope to replicate the results on other algorithms in the near future so as to allow comparison of PHC/SG+KL with other methods.

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 (`5655, 8870, Test02, Test03, Test04` and `Test06`) the improvement is very small; for others (`19ks, PrimGA1, Test05` and $G(1000, 0.0025)$) the improvement is small, but significant; and for the remaining three graphs (`PrimGA2, bm1`, and $U(1000, 0.04)$) the improvement is substantial. There is no obvious correlation between the degree of relative superiority of the PHC/SG+KL algorithm and the cardinality or average degree of the graphs in question.

For hybrid algorithms that involve the KL algorithm, the following question naturally arises: How much work is the KL part doing? In Table 3, an approximately time-equated comparison of the KL and PHC/SG algorithms is presented. (PHC/SG is the PHC/SG+KL algorithm without the KL refinement post-pass in Step 5. The data in Table 3 were derived from the same experimental tests described in Table 2, so the KL algorithm is given about 5% more time than the PHC/SG algorithm.) Perhaps surprisingly, the PHC/SG algorithm still manages to outperform the KL algorithm on five of the graphs (`8870`, `PrimGA1`, `PrimGA2`, `bm1`, and $U(1000, 0.04)$), substantially in some cases.

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. Furthermore, we can as yet offer no analysis that would

[6]Unfortunately a direct comparison with other algorithms on the MCNC 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 [1, 8, 10]. 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.

	KL		PHC/SG		
Graph	avg min cut-set size	σ	avg min cut-set size	σ	% improvement over KL
19ks	1020.8	33.5	1093.2	101.4	-7.1%
5655	603.2	4.3	612.8	2.9	-1.6%
8870	52.8	1.3	52.0	0.0	1.5%
PrimGA1	235.6	15.6	230.0	4.2	2.4%
PrimGA2	1051.6	77.5	751.8	31.1	28.5%
Test02	1172.8	11.9	1209.0	15.9	-3.1%
Test03	821.8	8.1	827.6	9.5	-0.7%
Test04	1191.2	2.2	1218.4	11.7	-2.3%
Test05	1887.4	26.4	1968.6	41.1	-4.3%
Test06	1194.4	3.6	1222.6	12.5	-2.4%
bm1	240.4	14.5	217.4	4.0	9.6%
$G(1000, 0.0025)$	98.2	3.0	98.4	1.1	-0.2%
$U(1000, 0.04)$	28.6	5.7	4.2	0.4	85.3%

Table 3: Kernighan-Lin and PHC/SG: an empirical comparison.

indicate why PHC/SG+KL is much better than KL on some graphs but not on others. Our agenda for future work therefore 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 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, though its performance remains to be seen.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

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

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

[15] 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.

[16] 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.

[17] 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.