MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

A Recursive Coalescing Method for Bisecting Graphs

Bryan Mazlish, Stuart Shieber, Joe Marks

TR94-10 December 1994

Abstract

We present an extension to a hybrid graph-bisection algorithm developed by Bui et al. that uses vertex coalescing and the Kernighan-Lin variable-depth algorithm to minimize the size of the cut set. In the original heuristic technique, one iteration of vertex coalescing is used to improve the performance of the Kernighan-Lin algorithm. We show that by performing vertex coalescing recursively, substantially greater improvements can be achieved for standard random graphs of average degree in the range [2.0,5.0]. Keywords: algorithms, combinatorial problems, design of algorithms, empirical analysis of algorithms, heuristic search.

Publication History:–

1. First printing, TR-94-10, June 1994

1 Introduction

In a graph-bisection problem, we are given a graph $G = (V, E)$, such that $|V| = 2n$. The goal is to find a pair $\langle A, B \rangle$ of independent subsets of V, where $|A| = |B| = n$, such that the number of edges with endpoints in both A and B is minimized. Since the graph-bisection problem is NP-hard (Garey, Johnson, and Stockmeyer, 1976), it is reasonable to assume that there is no efficient way to solve such problems exactly. A number of heuristic techniques have been proposed for bisecting graphs (Kernighan and Lin, 1970; Fiduccia and Mattheyses, 1982; Bui et al., 1989; Goldberg and Burnstein, 1983; Krishnamurthy, 1984). The long-standing champion, as evidenced by extensive empirical testing (Johnson et al., 1989), is the Kernighan-Lin (KL) variable-depth algorithm (Kernighan and Lin, 1970), presented in Figure 1. The details of this algorithm are not important for the purposes of the present discussion. The crucial characteristic of the algorithm, in addition to its good performance, is that it starts with an initial bisection (chosen randomly) and iteratively improves it to form the final bisection.

Recently, a variation on the algorithm, which uses *vertex coalescing*, has yielded significantly better solutions for graphs of small average degree (in the range $[2.0, 4.0]$) (Bui et al., 1987; Bui et al., 1989).[1] In this paper, we extend this coalescing idea, improving it nearly as much as it improves upon the original Kernighan-Lin algorithm.

[1] According to Johnson et al. (1989), "most interesting applications involve graphs with a low average degree," so the focus in this paper is on graphs with average degree in the range $[2.0, 5.0]$.

```
KL(G, ⟨A, B⟩)
{
  int i, k, swap_val, cumulative_swap_val_total, cumulative_swap_vals[];

  repeat
  {
    ⟨A_tmp, B_tmp⟩ ← ⟨A, B⟩;
    i ← 1;
    cumulative_swap_val_total ← 0;
    while (|A_tmp| > 0) do
    {
      (a_i, b_i, swap_val) ← best_swap(G, ⟨A_tmp, B_tmp⟩);
        /* best_swap() returns the best pair of vertices in A_tmp and B_tmp
        to swap, and the change in cut-set size that results from the swap. */
      cumulative_swap_val_total ← cumulative_swap_val_total + swap_val;
      cumulative_swap_vals[i] ← cumulative_swap_val_total;
      ⟨A_tmp, B_tmp⟩ ← ⟨A_tmp − {a_i}, B_tmp − {b_i}⟩;
      i ← i + 1;
    }
    k ← index_of_max_positive_val(cumulative_swap_vals);
      /* k = 0 if all entries in cumulative_swap_vals are ≤ 0. */
    if k > 0 then
      ⟨A, B⟩ ← (⟨A − {a_1, ..., a_k} + {b_1, ..., b_k}, B − {b_1, ..., b_k} + {a_1, ..., a_k}⟩);
  }
  until k = 0;
  return(⟨A, B⟩);
}
```

Figure 1: The Kernighan-Lin graph-bisection algorithm.

```
C(base)(G)
{
  M ← maximal_matching(G);
  G' ← coalesce(M, G);
  ⟨R, S⟩ ← random_bisection(G');
  ⟨A', B'⟩ ← base(G', ⟨R, S⟩);
  ⟨A, B⟩ ← uncoalesce_and_rebalance(⟨A', B'⟩);
  return(base(G, ⟨A, B⟩));
}
```

Figure 2: The vertex-coalescing technique.

2 Coalescing and Recursive Coalescing

Bui et al. proposed the algorithm described in Figure 2 for improving the performance of any given bisection algorithm (the *base algorithm*) that works by improving an initial bisection of a graph; the Kernighan-Lin algorithm is of this type. First, a maximal matching $E_M \subseteq E$ of G is computed: by definition, no two edges in E_M share a vertex and all edges not in E_M share a vertex with an edge in E_M. All pairs of vertices joined by edges in E_M are then coalesced to form a new smaller graph $G' = (V', E')$, typically of higher average degree. From a random bisection $\langle R, S\rangle$ of G', a better bisection $\langle A', B'\rangle$ is computed using the base algorithm. The edges in G' are then uncoalesced; applying this operation to $\langle A', B'\rangle$ induces a partition of G that may not be balanced, so a rebalancing operation may be necessary to produce a bisection $\langle A, B\rangle$ of G. (The vertices moved in the rebalancing operation—typically very few—are chosen randomly.) This bisection is then subjected to one more application of the base algorithm. We will use the functional notation $C(base)$ to refer to the algorithm generated by augmenting a base algorithm *base* with the coalescing method.

Bui et al. concluded that the reason coalescing helps is that previously reported heuristic techniques (and the Kernighan-Lin algorithm in particular) tend to work better on graphs of higher degree. However, if this idea works for the original graph G, then the coalescing paradigm should likewise help

```
RC(base)(G)
{
  if G.E = {} then
  {
    ⟨R, S⟩ ← random_bisection(G);
    return(base(G, ⟨R, S⟩));
  }
  M ← maximal_matching(G);
  G' ← coalesce(M, G);
  ⟨A', B'⟩ ← RC(base)(G');
  ⟨A, B⟩ ← uncoalesce_and_rebalance(⟨A', B'⟩);
  return(base(G, ⟨A, B⟩));
}
```

Figure 3: The recursive vertex-coalescing technique.

for the coalesced graph, G', which will usually have higher average degree than G. By applying this idea recursively we can use the coalescing algorithm on increasingly smaller graphs of increasingly higher average degree until we cannot coalesce the graph any further (see Figure 3). The base case of the recursion occurs when the edge set E of G is empty, in which case a random bisection is passed to the base algorithm. The final bisection of each graph yields an initial bisection for the graph one level higher in the recursion. We will use the notation $RC(base)$ to refer to the augmentation of a base algorithm *base* with this recursive coalescing method.

3 Empirical Testing of the Methods

We implemented the original Kernighan-Lin algorithm (KL), and the coalesced ($C(KL)$) and recursively coalesced ($RC(KL)$) variants, and compared their performance on a set of random graphs. We used the standard $G(m, p)$ random-graph model for our tests. In this model, there are m vertices in the graph, and edges are independently placed between each pair of vertices with probability p. Studies by Bui et al. and Johnson et al. have shown that tests on these graphs give a good indication of likely performance on other kinds of random graphs (Bui et al., 1989; Johnson et al., 1989).

Our test suite contained 25 graphs constructed for five different values each of m and p. The sizes of the graphs increase on a logarithmic scale to give $|V| = 124$, 250, 500, 1000, and 2000. For each value of $|V|$ we created graphs by choosing p so that the expected average degree of the graph would be 2.5, 3.5, 5, 10, and 20. (The actual average degree of each test graph was within 0.05 of the target value.) Each algorithm was run 1000 times on every graph, using different random bisections for each run. Running times for any particular graph were within a factor of two for all three algorithms, as would be expected.[2]

For graphs with average degree in the range $[2.0, 5.0]$, $RC(KL)$ dominates $C(KL)$, which dominates KL. The chart in Figure 4 shows the change in average solution quality for graphs of degree 2.5. (Following the convention adopted by Johnson et al. (Johnson et al., 1989), results are reported in terms of the best cut found by any of the algorithms for the graphs in question.)

A different perspective on the relative performance of the algorithms is shown in Figure 5. The chart in this figure shows the distributions of solutions found by the algorithms for graphs with 1000 vertices and average degree 2.5. The improvement of $RC(KL)$ over $C(KL)$ as captured in the relative shift of the distribution to the left is approximately the same as that of $C(KL)$ relative to KL.

As the graph size increases, however, the comparative advantage of the

[2]For each combination of vertex-set size and average degree there are a large nuber of graphs that can be generated. We conducted a number of tests to determine how much one graph can bias algorithmic performance. These tests verified that algorithmic performance has very low variability across standard random graphs with the same average degree and vertex-set size. We can thus use one random graph for each combination of vertex-set size and average degree without sacrificing accuracy in our results.

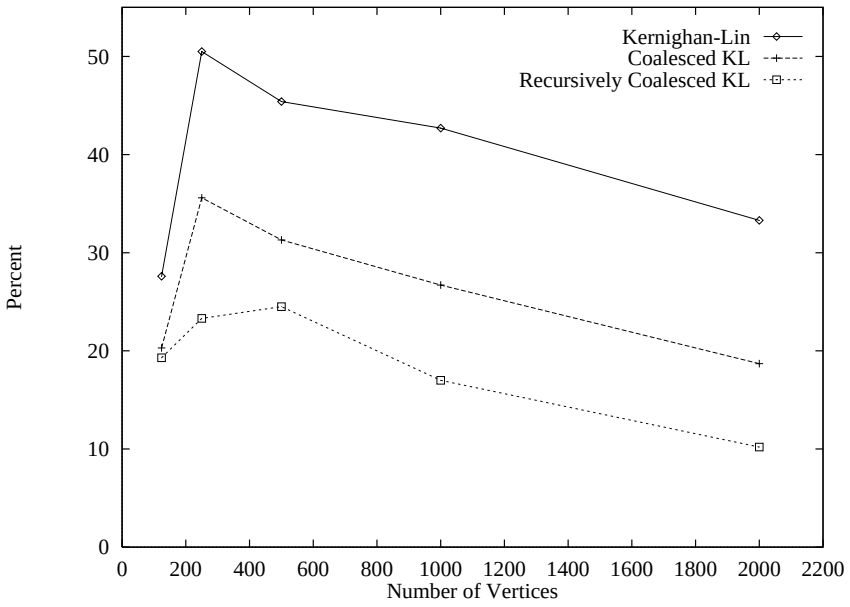

Figure 4: Performance of KL, $C(KL)$, and $RC(KL)$ on graphs of varying numbers of vertices with average degree 2.5.

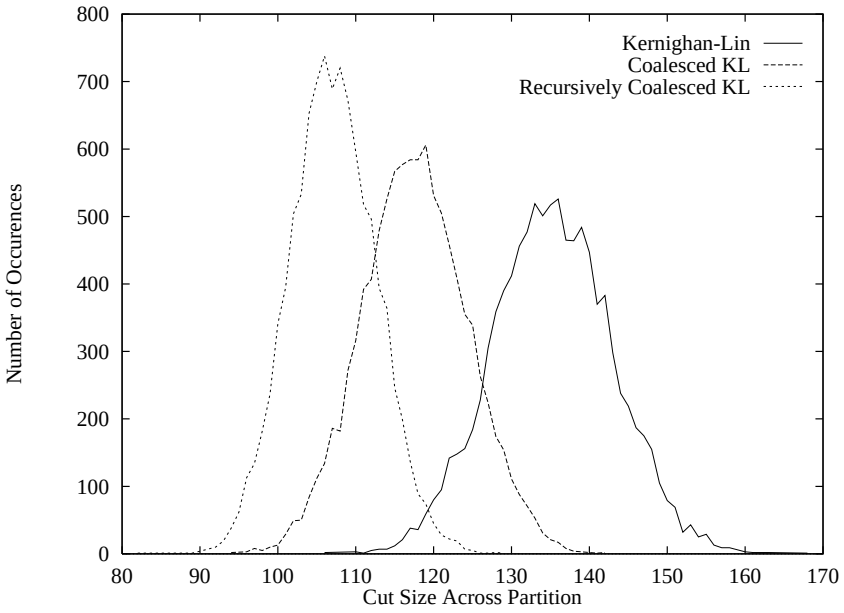

Figure 5: Distributions for KL, $C(KL)$, and $RC(KL)$ of cut-size found in 10,000 trials for a single $G(1000, 2.5)$ graph.

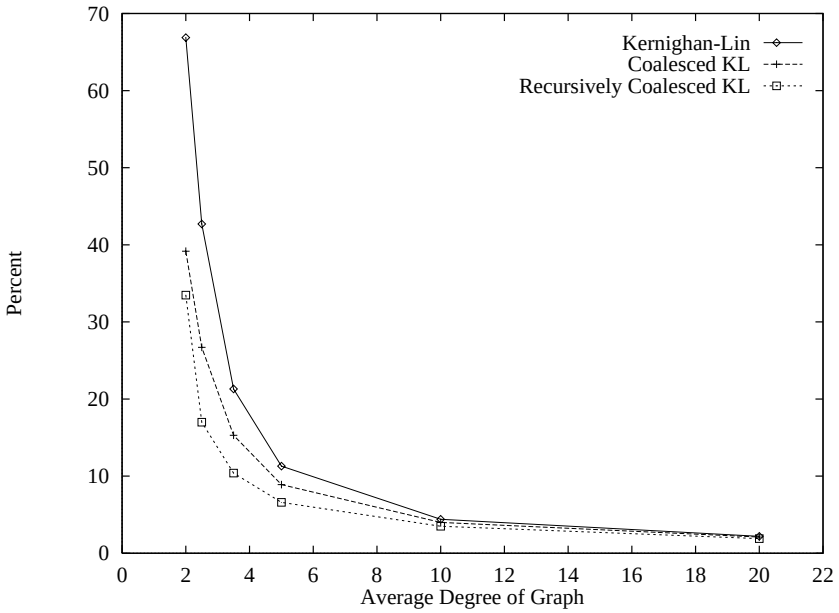

Figure 6: Performance KL, $C(KL)$, and $RC(KL)$ for 1000-vertex graphs of varying average degree.

$RC(KL)$ algorithm decreases. The benefits from $RC(KL)$ (and $C(KL)$) can only be realized on graphs of small average degree (less than 5.0). When the average degree increases above 5.0, the relative effectiveness of $RC(KL)$ decreases dramatically. The chart in Figure 6 shows how average solution quality varies with average degree for graphs of 1000 vertices. (Graphs with average degree less than 2.0 are so sparse that very good bisections are found easily by all of the algorithms.)[3]

Since all three of the algorithms are not completely deterministic — they

[3]Although Figure 6 appears to indicate that absolute algorithmic performance decreases as the vertex-set size decreases, absolute algorithmic performance is better for graphs of lower degree. The confusion arises in the normalization of the average algorithmic performances. For graphs of low average degree a small change in cut size has a large affect on the ratio of average cut size to best cut size since the best cut size found is quite small (54 for the $G(1000, 2.0)$ graph). This skews the apparent absolute algorithmic performance. One should thus restrict interpretation of Figure 6 to relative algorithmic performance.

have an arbitrary aspect in the choice of initial bisection and, in the case of the coalescing algorithms, the choice of maximal match — they allow for a trade-off between time and solution quality by conducting a search among various instantiations of the nondeterministic choices. For instance, one can generate independent solutions from the space of nondeterministic possibilities and choose the best; this gives rise to a random-generate-and-test search regime. (Alternatively, a more directed type of search, such as hill climbing or simulated annealing might be entertained, though we do not pursue these possibilities here.) Figure 7 shows that a random-generate-and-test search in the spaces defined by the three algorithms preserves the relative advantage of $RC(KL)$ over $C(KL)$ and KL. In the figure, performance was averaged over 50 runs of each algorithm on the same $G(1000, 2.5)$ graph. Error bars show standard deviations for the averages.

Even when running time is taken into account, $RC(KL)$ still has a substantial edge. Figure 8 shows relative performance of a random-generate-and-test search normalized to equalize the running times of all algorithms. Thus, for any given running time, the number of iterations allotted to $RC(KL)$ was less than the number allotted to $C(KL)$, which in turn was allotted fewer than KL. As the figure shows, even normalized in this way, the ranking of $RC(KL)$ over $C(KL)$ over KL is preserved. Given a fixed amount of time to search for solutions, the $RC(KL)$ algorithm still dominates the other two.

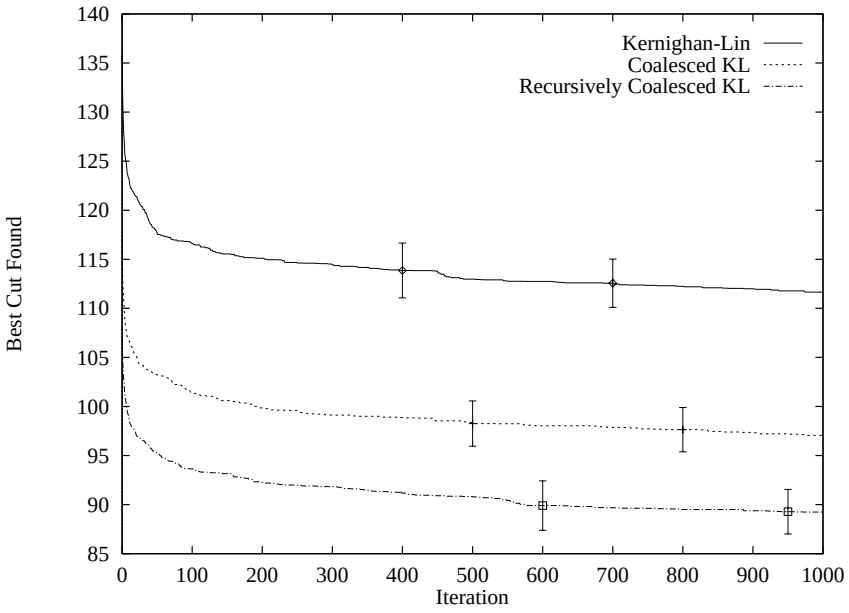

Figure 7: Iterated algorithm performance on a $G(1000, 2.5)$ graph as a function of number of iterations.

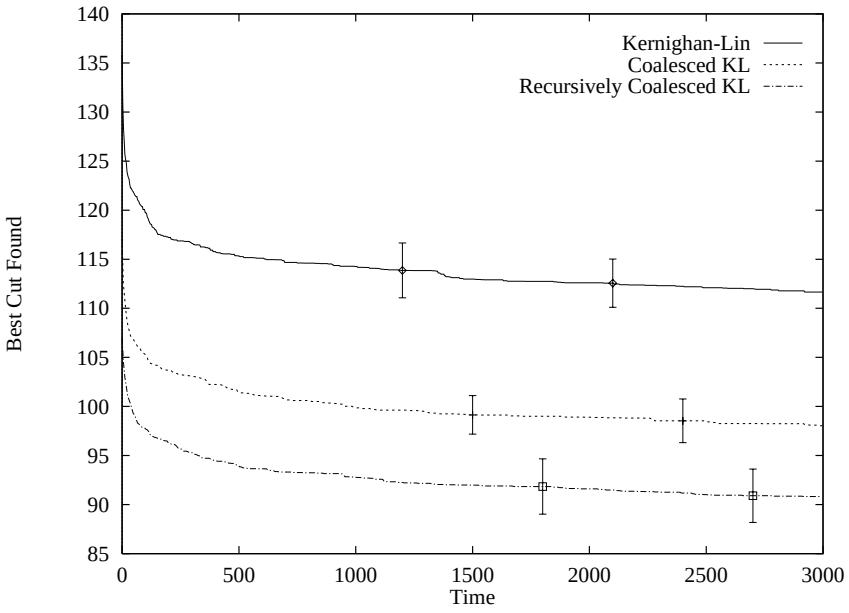

Figure 8: Iterated algorithm performance on a $G(1000, 2.5)$ graph as a function of amount of running time (in seconds on a Digital 3000/400 AXP workstation).

4 Conclusion

For random $G(m,p)$ graphs with average degree in the range $[2.0, 5.0]$, the $RC(KL)$ algorithm — a recursive variant of Bui et al.'s $C(base)$ technique applied to Kernighan and Lin's KL algorithm — has significantly better average performance, both on a per-iteration and per-unit-time basis, than the $C(KL)$ and KL algorithms.

References

Bui, T., S. Chaudhuri, F. Leighton, and M. Sipser. 1987. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191.

Bui, T., C. Heigham, C. Jones, and T. Leighton. 1989. 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.

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

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

Goldberg, M. K. and M. Burnstein. 1983. Heuristic improvement technique for bisection of VLSI networks. In *Proceedings of the IEEE International Conference on Computer Design*, pages 122–125, Port Chester, N.Y.

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

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

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