The dynamic locking heuristic - a new graph partitioning algorithm

Achim G. Hoffmann
University of New South Wales
School of Computer Science and Engineering
P.O. Box 1, Sydney, NSW 2033, Australia
E-mail: achim@cs.unsw.oz.au

Abstract

In layout design finding efficiently good solutions to the hypergraph bipartitioning problem is of great importance. This paper introduces a new algorithm, the *dynamic locking algorithm*, which has empirically shown a significant improvement in the partitioning result compared to known algorithms.

1 Introduction

The problem of graph partitioning occurs in many domains of significant practical importance and economic impact. Among them is the design of VLSI chips. Unfortunately, graph bipartitioning has been proved to be NP-hard [4]. However, fast heuristic algorithms for the problem have been developed [6, 3]. One of the most popular heuristics among them is the heuristic of Fiduccia & Mattheyses [3] (in short FM) which is a fastened form of the heuristic of Kernighan & Lin [6] (in short KL). Krishnamurthy [7] proposed a class of possible improvements of the FM heuristic. Dunlop and Kernighan [2] developed a sophisticated application of FM for recursively partitioning a circuit into many parts for the placement problem in circuit design. Bui et al. [1] suggested to build clusters of nodes before bisecting the graph in order to improve the performance of the Kernighan-Lin heuristic. This paper introduces a new hypergraph bipartitioning algorithm. The new algorithm, the *dynamic locking algorithm* (in short DLA), has empirically shown a significant improvement in the resulting cutsize over FM. The dynamic locking algorithm has a linear time complexity.

The paper is organized as follows. The next section provides a more technical treatment of the problem of hypergraph bipartitioning and of previous work. Section 3 introduces the new *dynamic locking algorithm*. In section 4 an empirical comparison of the new algorithm to the FM algorithm is given. Section 5 contains concluding remarks.

2 Previous Work

Netlists can be described as hypergraphs, i.e. graphs where edges may contain more than two nodes. The following problem will be called hypergraph bipartitioning:

Given: An undirected hypergraph $G = (V, E)$. An hyperedge $e \in E$ is a subset of V.
Problem: Find two disjoint, almost equally sized subsets V_1, V_2 of V such that the number of edges that connect nodes in different partitions is minimal. That is: Find $V_1, V_2 \subset V$, where $V_1 = (V \setminus V_2) \ \wedge (|V_1| \leq |V_2| \leq |V_1| + 1$, such that the set of shared nets

$$|\{e | e \in E \wedge (\exists n_1, n_2 \in e \wedge n_1 \in V_1 \wedge n_2 \in V_2\}|$$

has its minimum over all partitions.
Thus, the problem of graph bipartitioning, i.e. where each edge has only two nodes is a special case of the hypergraph bipartitioning problem as stated above.

2.1 The KL heuristic

The Kernighan-Lin heuristic (in short KL) starts from a given initial bipartitioning of the (hyper-)graph and tries to find a near-optimal solution by stepwise modifying the initial partitioning. The initial partitioning may be randomly generated or obtained by other means. To improve the initial partitioning it is assumed that some nodes are not in their 'right' partition. I.e. if these nodes would be in the opposite partition, the partitioning would have its optimum. Consequently, the Kernighan-Lin heuristic tries to identify in each partition one of these nodes in order to swap them. KL's approach to identify a pair of nodes to be swapped is as follows: For each pair of nodes located

in different partitions KL determines by how many nets the partitioning would improve after swapping. The heuristic chooses that pair of nodes for swapping that has the highest gain. For n nodes this results in $((\frac{n}{2})^2/2) = \frac{n^2}{8}$ gain computations for determining the best pair to be swapped. Therefore, the Kernighan-Lin heuristic has a quadratic time complexity $O(n^2)$.

After a node has been moved it is not considered for another swap. This suits the idea of having initially some nodes in the 'wrong' partition. Once nodes are swapped it is assumed that they moved to their 'right' partitions.

2.2 The FM heuristic

The Fiduccia-Mattheyses heuristic is in its basic idea very similar to Kernighan-Lin. The FM heuristic has fastened KL by abandoning the idea of interchanging two nodes in a single step. Instead, Fiduccia and Mattheyses proposed to move a single node at a time [3]. Consequently, only for each single node its gain for moving to the opposite partition has to be computed. The node with the highest gain is selected and moved to the opposite partition. Of course, it has to be avoided that all nodes are moved to one partition - although the cutsize would have its minimum of 0 ! For that purpose FM uses a balancing criterion, which requires both partitions to contain a prespecified minimal number of nodes. Like KL, FM allows each node to move only once.

Note that nodes with a negative gain for moving may still be moved to overcome some local optima. After each node has been moved once, the best partitioning which emerged through the entire moving process is determined as the result.

Let us call the process the *improvement process*. The result of the improvement process is then used as the initial partitioning for further improvement processes. The entire process terminates, when the *improvement process* yields no further improvement any more.

Both algorithms - KL as well as FM - consist of two cascaded loops. The *inner loop* covers the *improvement process*. The *outer loop* is called the *iterative improvement loop* and contains the inner loop. I.e. the iterative improvement loop starts with some initial partitioning and feeds the partitioning to the inner loop. The best found partitioning returned from the inner loop is fed back to another run through the inner loop until no further improvement is achieved.

3 The new approach

To motivate the new algorithm problems with the known algorithms are discussed first.

3.1 Weaknesses of the KL and FM

Both KL and FM move each node only once through its inner loop to avoid cycles in moving the nodes. Unfortunately, this strategy may cause the algorithm to stay in some local optimum. Since, instead of moving a node N back to its original partition if the respective gain indicates that move, KL and FM move other nodes to their opposite partition. See figure 1. This may result in an even worse situation: The gain of moving N back to its original partition which may actually be its 'right' partition, may even decrease for the following run through the inner loop. Thus, the fact that N has been locked during the last run through the inner loop, cannot always be compensated by the following run through the inner loop. Experiments have shown that in practice this is unfortunately very often a problem.

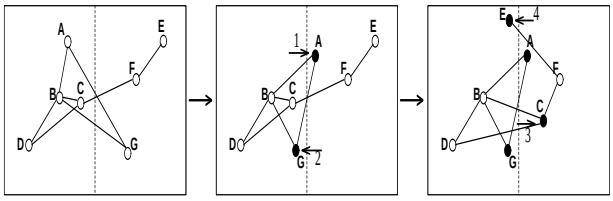

Figure 1: An example where the idea of moving each node only once results in a poor partioning result. The shown sequence of improvement attempts of the initial partitioning would be produced by the FM algorithm with a maximum of 4 nodes allowed in each partition. The filled nodes are not allowed to be moved again.

3.2 The dynamic locking algorithm

The *dynamic locking algorithm* (in short DLA) attacks the problem discussed above. Basically, DLA allows to move nodes between partitions back and forth.

In order to avoid cycles in moving nodes, DLA locks nodes immediately after they have been moved. The nodes stay locked as long as there is no promising reason that they should be moved again. When a promising reason emerges, then a node will be unlocked and becomes eligible for another move to its opposite partition. A locked node should possibly be moved another time if some or even all of its neighbors have been moved to the opposite partition. This may indicate that the current partition of a node is possibly not its 'right' partition.

DLA checks after each move of a node N from partition A to B the topological neighbors of N. All neighbors of N are unlocked which are located in partition A. Thus, a node may be moved from A to B, even if it had been moved from B to A at an earlier stage.

Figure 2 shows how DLA overcomes the weakness of FM as illustrated in figure 1.

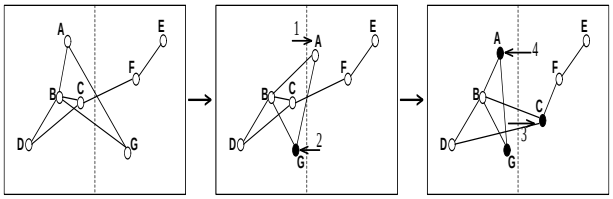

Figure 2: The shown improvement process would be generated by the new *dynamic locking algorithm.* (Only the unfilled nodes are allowed to be moved.) This would solve the problem of FM as shown in the previous figure. For this simple hypergraph the KL algorithm would solve the problem as well by choosing the best *pair* of nodes in opposite partitions for swapping. But the KL strategy does not suffice for the general case.

DLA avoids an infinite moving of nodes by imposing a maximum number of moves per node. Experiments have shown that a maximum number of about 10 appears to be a good balance between achieving a near optimal solution and having short runtimes of the algorithm.

Figure 3 shows the skeleton of the *dynamic locking algorithm* in pseudo code. In order to implement this strategy efficiently each node has a 'lock'-flag associated as well as a counter of the number of moves performed so far. The 'lock'-flag is maintained in the procedure 'delock_all_eligible_nodes' in the pseudo code of figure 3. Like the Fiduccia-Mattheyses algorithm, DLA takes the best solution of one run through its inner loop as the initial partitioning for the next run through the inner loop. The entire process is tried a few times with different randomly generated initial partitionings.

```
/* begin of  iterative improvement loop */
do
    /* begin of  inner loop */
    while admissable_move_is_available do
        determine_admissable_node_with_highest_gain n
        move_node n from p1 to p2
        lock_node n
        delock_all_neighbors of n in p1
        update_all_gains
        if (current_cutsize < local_best_cutsize)
          save_current_partitioning_as_local_best_partitioning
        endif
    endwhile
    /* end of  inner loop */
    if (local_best_cutsize < best_loop_cutsize)
        save_local_best_partitioning_as_best_loop_partitioning
    endif
while (best_loop_partition_has_been_improved)
/* end of  iterative improvement loop */
```

Figure 3: The new *Dynamic Locking Algorithm* in pseudo code.

4 Empirical comparison

The new *Dynamic Locking Algorithm* has been implemented in C and experimentally compared to a C implementation of the Fiduccia & Mattheyses [3] algorithm. FM has been chosen for comparison because it is known to be much faster than KL and still achieves usually results which are not much worse than the results by KL. The netlists used for the performance comparisons of DLA and FM contained between 250 and 10,000 nodes. The number of nets were the same as the number of nodes in each case. Each net had between 2 and n pins. n varied between 2 and 10. The number of nodes per net has been equally distributed between 2 and n. All nodes had the same probability to be connected by a particular net.

In the experimental comparisons DLA achieved up to 35% reduced cutsizes compared to FM for similar runtimes. In the following figures some of the obtained results are plotted. Figure 4 shows the results for hypergraphs with 1,000 nodes. The following figures 5 and 6 show the results for hypergraphs with 10,000 nodes. The cutsize comparisons in the figures have been done on a basis where the consumed runtimes of the two algorithms were kept comparable. I.e. since DLA moves each node up to 10 times back and forth it is to be expected that the overall running time for improving one initial random partitioning takes roughly 10 times as long as FM which moves each node only once (left diagram of figure 5). Interestingly, the relative performance improvement of DLA compared to

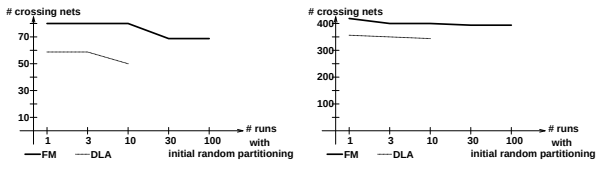

Figure 4: **Left:** Obtained partitionings for 1,000 nodes and 2 nodes per net. DLA achieved a 30-35% better result than FM. **Right:** Obtained partitionings for 1,000 nodes and up to 5 nodes per net. DLA achieved a 10-15% better result than FM.

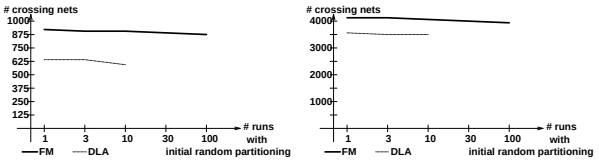

Figure 5: **Left:** Obtained partitionings for 10,000 nodes and 2 nodes per net. DLA achieved a 30-35% better result than FM. **Right:** Obtained partitionings for 10,000 nodes and up to 5 nodes per net. DLA achieved a 10-15% better result than FM.

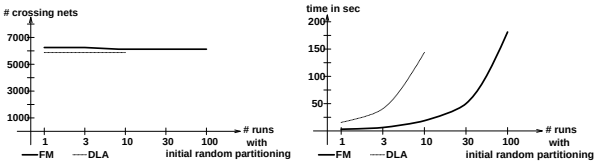

Figure 6: **Left:** Obtained partitionings for 10,000 nodes and 10 nodes per net. DLA achieved a 3-5% better result than FM. **Right:** The consumed computing time on a SPARC I ($\approx$ 12 MIPS) for both algorithms for the partitioning of the 1,000 nodes hypergraph with up to 5 nodes per net. Since the number of initial random partitionings on the x-axis is in a logarithmic scale the computing time grows roughly linear. The relation of the required computing time of the two algorithms is approximately the same for all other compared problems.

FM is almost independent of the number of nodes in a hypergraph. It rather depends strongly on the number of nodes per net. The percentage of cutsize reduction obtained by DLA compared to FM decreases when the number of pins per net increases. This dependency is quiet plausible since the more nets the average net contains the more nets will be across the partitions in the optimal solution. When the relative difference of the number of crossing nets between an initial random partitioning and the optimal partitioning decreases, the performance difference of approximating algorithms will usually decrease as well.

While DLA shows its strongest benefits with a cut-number reduction of up to 35%, for 2 nodes per net, DLA still provided for an average number of 6 nodes per net an improvement of 3 to 5% compared to FM. It should be noted that for many applications the average number of nodes per net is rather 3 than 6. Thus, the improvement of DLA compared to FM is likely to lie between 10% and 20% for many applications in layout design.

5 Conclusions and future work

This paper introduced the *dynamic locking algorithm* (DLA), a new algorithm for hypergraph bipartitioning. DLA has experimentally shown significantly better results (up to 35%) than the classical algorithm of Fiduccia and Mattheyses.

Future research will be concerned with enhancing DLA by initial clustering in order to treat heavily connected nodes as a single node in a (hyper-)graph. Also, the dynamic locking algorithm will be generalized in order to be used for multi-partitioning. Multi-partitioning has been shown to produce superior performance for the placement problem in VLSI layout design as opposed to recursive bipartitioning, see e.g. in [5, 8].

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* 26^{th} *Design Automation Conference*, pages 775–778, 1989.

[2] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Transactions on Computer-Aided Design*, 1(CAD-4):92–98, January 1985.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the* 19^{th} *Design Automation Conference*, pages 175–181, 1982.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, San Francisco, CA, 1979.

[5] A. G. Hoffmann. Towards optimizing global Min-Cut partitioning. In *Proceedings of the* 2^{nd} *European Design Automation Conference*, pages 167–171. IEEE, 1991.

[6] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(2):291–297, February 1970.

[7] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, C-33(5):438–446, May 1984.

[8] G. Vijayan. Min-cost partitioning on a tree structure and applications. In *Proceedings of the* 26^{th} *Design Automation Conference*, pages 771–774, 1989.