

Evaluating Iterative Improvement Heuristics for Bigraph Crossing Minimization

Matthias Stallmann Franc Brglez Debabrata Ghosh
CBL (Collaborative Benchmarking Lab), Dept. of Computer Science, Box 7550
NC State University, Raleigh, NC 27695, USA
<http://www.cbl.ncsu.edu/>

Abstract – *The bigraph crossing problem, embedding the two node sets of a bipartite graph $G = (V_0, V_1, E)$ along two parallel lines so that edge crossings are minimized, has application to placement optimization for standard cells and other technologies. Iterative improvement heuristics involve repeated application of some transformation on an existing feasible solution to obtain better feasible solutions. Typically an increase in the number of iterations, and therefore execution time, implies an improvement in solution quality. We investigate tradeoffs between execution time and solution quality in order to establish the best heuristic for any given time budget. Our experiments show some clear trends for a scalable class of graphs based on actual circuits. These trends, based on statistically significant samples of each of several graph sizes, suggest promising directions for development of better heuristics.*

Keywords: wire crossing minimization, iterative improvement, design of experiments, circuit equivalence classes.

1 INTRODUCTION

The minimization of the crossing number in a specific graph embedding has often been motivated by factors such as (1) improving the appearance of a graph drawing [1], and (2) reducing the wiring congestion and crosstalk in VLSI circuits, which in turn may reduce the total wire length and the layout area [2, 3, 4]. In fact we have been able to show remarkable correlation between crossing number and wire length obtained by routings based on various placements of VLSI circuits [5]. The *bigraph crossing* problem is defined formally as follows [6]: Let a bipartite graph (bigraph) $G = (V_0, V_1, E)$ be embedded in the plane so that the nodes in V_i occupy distinct positions on the line $y = i$ and the edges are straight lines. For a specific embedding $f(G)$, the *crossing number* $C_f(G)$ is the number of line intersections induced by f . This depends only on the permutation of V_i along $y = i$ and not on specific x-coordinates. The (bigraph) crossing number $C(G) = \min_f C_f(G)$. Garey and Johnson [7] proved that it is NP-hard to compute $C(G)$, hence the need to develop heuristics that find f for which $C_f(G)$ is as close as possible to $C(G)$.

Bigraph crossing is an important special case of multi-row placement problems that arise, for example, in standard cell layout [8]. Nodes of V_0 represent cells while nodes of V_1 represent interconnecting nets. Figure 1(a) shows a bigraph representation of a circuit with cells a, b, c, d, e, f and nets A, B, C, D , where A connects to cells a, b, d, f , etc. The initial placement shown at the top has 22 crossings and any routing of it would have much congestion and total wire length. The bottom of the figure shows an optimal solution with only 7 crossings, a decrease of more than a factor of 3, resulting in a similar decrease in wire length (and other

statistics such as area) for routings.

Iterative improvement heuristics begin with an *initial solution*, which may be random or computed by another heuristic, and perform repeated transformations on the current solution in order to obtain a solution with the lowest possible cost.

One of the iterative improvement heuristics examined here is based on the median and barycenter heuristics for a variant of bigraph crossing in which the order of one node set is fixed (see [1]). If V_0 is fixed, for example, new positions for V_1 nodes are computed based on the average positions of their neighbors in V_0 (assuming the leftmost node is at $x = 0$ and nodes are unit distance apart). This average can be the mean (leading to the barycenter heuristics) or the median (yielding the median heuristic: if the number of neighbors is even, the smaller of the two medians is used).

These heuristics turn into iterative improvement if applied repeatedly, alternating the fixed node set between V_0 and V_1 . Each application of the median or barycenter heuristic is called a *pass*. A sequence of two passes, one with V_0 fixed, the other with V_1 fixed, is called an iteration.

Earlier experiments suggest that an additive mixture of median and barycenter performs better than either one by itself [9]. Consider using the top ordering of Figure 1(a) as an initial solution. Numbers above the V_1 nodes show the median and the mean position of its neighbors: the two values are added together and the nodes are sorted by that sum. The result of that first pass is shown at the top of Figure 1(b): the crossing number has been reduced to 18. A pass with V_1 fixed (using numbers below V_0 nodes to sort them) completes the first iteration and reduces the crossing number to 11. The bottom picture shows the result and also shows that this is a stable solution in the sense that the median/barycenter mix will no longer make any changes. While we have no proof that iterative improvement based on averaging will always converge in this way, our experiments have not yet found a counterexample.

We recently developed a new heuristic called *adaptive insertion* [9]. The main idea is that during every pass, each node of a given layer is inserted among others of the same layer in a position that minimizes new cost minus old cost. Details that are the result of repeated experimentation with many variations on this theme are described in Section 2. As with the averaging heuristics, two passes, one per layer, constitute an iteration. A pass of adaptive insertion may decrease or increase cost (nodes are forced to move even if their current position is better than any other). The best ordering encountered so far is saved.

Our aim is a close examination of the tradeoffs between the quick convergence of averaging heuristics and the slow but steady improvement yielded by adaptive insertion. For different sizes of graphs, how much execution time do we need to be able to spend in order to obtain better solutions with adaptive insertion than with averaging? Are there situations in which a mixture of the two strategies does better than either one by itself (in terms of solution quality ob-

This research was supported by contracts from the Semiconductor Research Corporation (94-DJ-553), SEMATECH (94-DJ-800), and DARPA/ARO (P-3316-EL/DAAH04-94-G-2080 and DAAG55-97-1-0345).

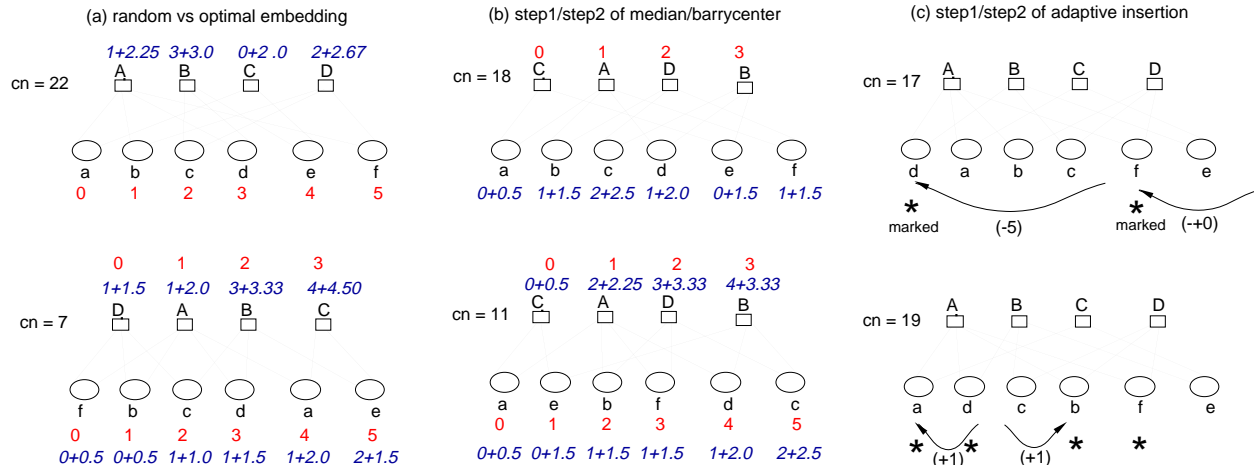


Fig. 1. An example of a bigraph used to illustrate the heuristics.

tainable in a specific amount of time)?

The rest of the paper is organized as follows. Section 2 gives a more detailed description of adaptive insertion and what combinations of heuristics are used in our experiments. Section 3 explains our experimental design. Section 4 gives an overview of our results. And Section 5 summarizes our work and poses some open questions.

2 HEURISTICS

To illustrate a single pass of adaptive insertion, consider rearranging V_0 of the top of Figure 1(a) by a sequence of insertions. The heuristic works from right to left (this is an arbitrary choice). First f finds its best position relative to the other nodes. Ties are broken by choosing the position closest to the current one, but nodes are not allowed to remain where they are. The best position turns out to be an insertion before e with no change in crossings. Now the right to left pass encounters f again (e is ignored in the interest of progressing farther to the left), but f is marked as having been inserted already. Node d is the next one to be inserted into its best position, before a . This insertion is equivalent to swapping d with each of c , b , and a in turn. Each swap can be evaluated independently [1]. Swapping d with c decreases crossing number by 3 because cD no longer crosses dA or dB and cB no longer crosses dA . Swaps of d with b and a each yield a decrease of 1 for a total decrease of 5. The top of Figure 1(c) illustrates the embedding after these first two insertions. The next node encountered is b , and its best insertion turns out to be to the right (we allow either direction), after c , yielding an *increase* of 1 (staying in place is not an option!) Finally a is forced to move and finds its best position before d . The final position after one pass of adaptive insertion is shown at the bottom of Figure 1(c).

On this small example the advantages of adaptive insertion are far from clear. In intermediate ordering with 17 crossings is abandoned in favor of the 19-crossing configuration at the end of the pass. If we continue with several additional passes on this example, it appears that good configurations are quickly undone by the heuristic's insistence on inserting even when cost is thereby increased. It takes 7 passes before an ordering with cost 11 is encountered, and even after more than 12000 passes no better solution is found. On large graphs however, the forced insertions cause adaptive insertion to explore solutions that are radically different from the initial ordering and this appears to give it a

major advantage over the averaging heuristics, which tend to get stuck in nearby stable configurations.

A blend of averaging and adaptive insertion bridges the gap between these two extremes by alternating iterations between adaptive insertion and the median/barycenter mix. On our example, the first iteration has two insertion passes ending with 19 and 24 crossings, respectively. The first pass of averaging finds an ordering with 9 crossings, and after only 6 iterations (ending with averaging again), an optimal solution was found.

Since it appears that a better initial solution leads to a better final solution with all the iterative improvement heuristics, our experiments use the best possible starting points we know how to compute, namely those obtained by *guided breadth-first search* as described in [9]. We use the treatment numbers of that paper, except that we let the averaging heuristic, **tr16**, run until it converges, usually less time than it takes to do the first few iterations of insertion, and we set the number of iterations for the heuristics that use insertion, **tr18** (pure adaptive insertion) and **tr19** (alternating iterations with the median/barycenter mix), so that every run takes roughly the same amount of CPU time, regardless of which of the two treatments is used.

As documented in [9], time for one pass of averaging is $O(n \log n)$ and insertion takes $O(n^2)$ (n is the number of edges, which, for our circuits, is linear in the number of nodes). These growth rates have been verified experimentally.

3 EXPERIMENTAL DESIGN

Our experimental setup is as described in [10]. Recall that our goal is to discover which heuristic will yield the best possible solution within a given time budget. Our answer is a function of that time budget and the number of edges in the graph. It may also depend on other factors such as the type of graph, but those issues are outside the scope of this investigation.

We start with a simple hypothesis: For most graphs and initial solutions, there are three time quantities t_0 , t_1 , and t_2 such that **tr16** (averaging) converges in CPU time t_0 , the best solution is obtained by **tr16** as long as the CPU time (budget) is $< t_1$, by **tr19** (the mixture of averaging and insertion) when time is $\geq t_1$ but $< t_2$, and by **tr18** (pure insertion) when time is $\geq t_2$.

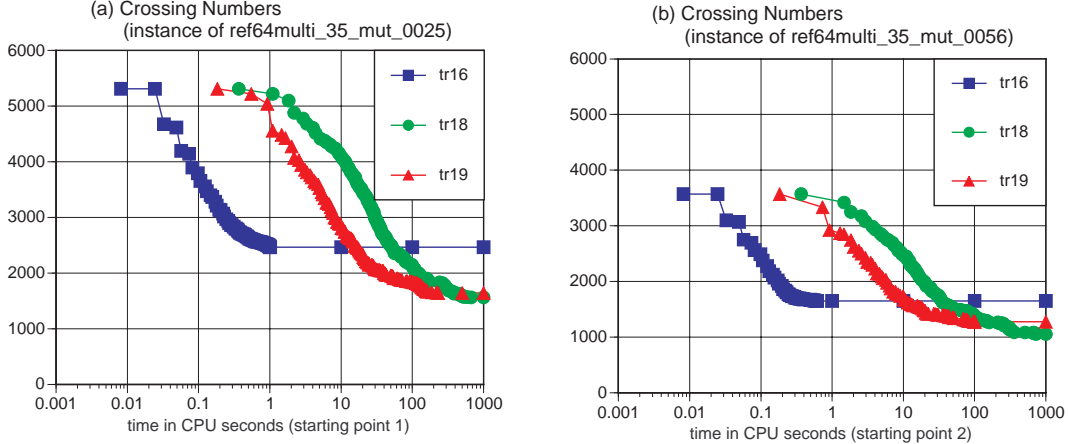


Fig. 2. Time versus solution quality for three heuristics on two different initial solutions.

The experiments are focused on a class of graphs based on a substructure commonly found in circuits. We start with a *reference graph* G_q that consists of two simple cycles joined to a *comb* (bigraph that can be embedded with no crossings) via a degree-2 cell node. The parameter q takes on values 32, 64, and 128 and each G_q has $8q + 5$ nodes and $8q + 6$ edges. For each G_q we generate 64 mutants, graphs that have the same number of nodes and edges as G_q , are connected, and preserve the invariant that each cell node has degree 2 (see [11] for more details). Thus we obtain a random sample of graphs sharing certain characteristics and can draw statistically significant conclusions about our experiments. Finally, for several of the mutants, we created *isomorphism classes*, graphs that differ only in the order in which their nodes are presented and the node names. This gave us opportunity to see the effect of the initial solution on our results.

Figure 2 illustrates the behavior of the three treatments on two different starting solutions for the same graph, a mutant of G_{64} with 518 edges. Observe that **tr16** converges in less than a second, it takes **tr19** more than 10 seconds to catch up to and ultimately do better than **tr16**, and **tr18** finally catches up and does better after more than 100 seconds. The better starting solution leads each of the heuristics to a better final solution, and allows the slower heuristics to catch up more quickly. The curves in these figures connect only those points where a heuristic found a solution better than previous ones. Both **tr18** and **tr19** exhibited much variation in cost from one iteration to the next.

The behavior illustrated in the figure conforms to our hypothesis with each of t_i an order of magnitude greater than the previous one. To validate our hypothesis and gather data about typical values of t_1 and t_2 (the value of t_0 was never large enough to be significant), we ran each treatment on each of the 64 mutants in each size class and recorded values of t_1 and t_2 for each mutant. We also picked a sample of 5 mutants in the the class based on G_{64} and generated 64 isomorphic copies of each to record t_1 and t_2 for different starting solutions. Runs on mutants based on the smaller graphs G_8 and G_{16} were also done, but results on these were too erratic: in many cases **tr16** came up with the best solution even if we allowed the other heuristics to run significantly longer than the imposed limit (about 1000 seconds). In other cases **tr18** or **tr19** did better than **tr16** almost immediately.

4 RESULTS

Figure 3 shows histograms of the values of t_1 and t_2 encountered during our experiments with the three mutant classes. The distribution of t_1 values, shown across the bottom, appears reasonably well behaved. But the three- to four-fold increase in average time when the size of the graph is only doubled is disconcerting. Much of this is due to the fact that the average time per iteration for **tr19**, like **tr18**, is quadratic in graph size, while being almost linear for **tr16**. The t_2 values are unpredictable and sometimes very large. The ≥ 2000 cases are all situations where **tr18** was unable to find a better solution than **tr19** in the allotted time (about 1000 seconds for G_{32} and G_{64} , 2000 seconds for G_{128}). How long that might take or whether it occurs at all is still an open question. Therefore we have not succeeded as yet in validating our hypothesis about the existence of t_2 . The histograms do hold out the hope that if t_2 exists at all, then it grows more slowly with the size of the graph than t_1 .

Still, the execution times at which **tr18** is better than **tr19** are too large to be scalable to current circuit sizes and the growth of t_1 suggests that even the range for which **tr19** is better will not scale well. This suggests that a mix that leans more toward averaging than **tr19** might be a good prospect, for example a heuristic that only does insertions when the averaging gets stuck, and hence only engages in quadratic behavior after a longer sequence of iterations of a simpler heuristic.

One interesting side benefit was the discovery that the initial solutions for one of the G_{64} mutants, the one with results depicted in Figure 2 had crossing numbers in two disjoint ranges, one clustered around 3500, another around 5000. Apparently the guided breadth-first search had fewer choices that depended on input order. For this mutant there was a clear correspondence between initial solution quality and t_1 (values of t_1 also exhibited two disjoint ranges that corresponded to those of initial crossing number, with the exception of one isomorph): a better initial solution meant a smaller t_1 . The same observation held for t_2 , but with more exceptions.

5 CONCLUSIONS

We have studied the tradeoffs between execution time and solution quality in a very limited domain. Three heuristics that represent two extremes and something in the middle have been applied to bigraph crossing minimization on a graphs of varying sizes drawn from a scalable class of graphs.

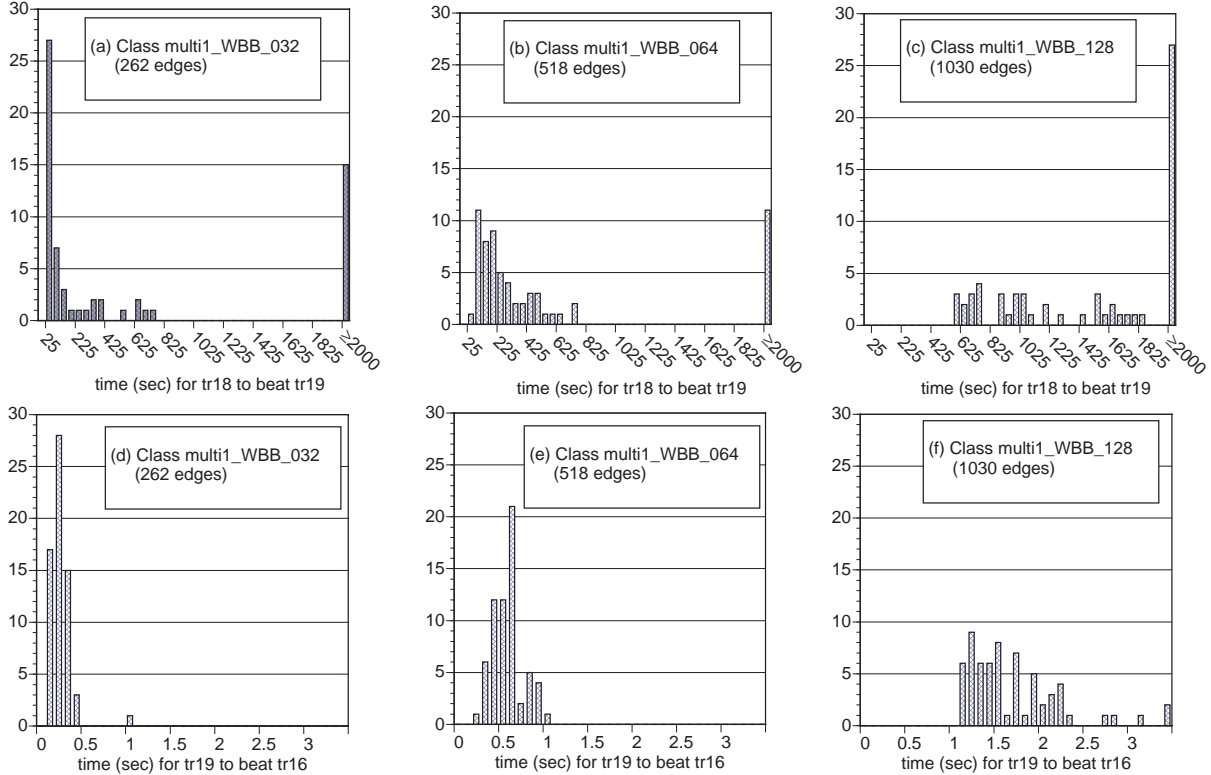


Fig. 3. Distribution of t_1 and t_2 values for graph classes of 3 different sizes.

The ability to work with many similar graphs of each size and to have characteristics that are invariant as the graphs increase in size (e.g. relationship between number of nodes and edges, degree of cell nodes, connectivity) has been critical to our results. More importantly, the presence of such a testbed allows us to do future work that can be related directly to these results. We have seen that the choice of heuristic as a function of available time can be made fairly predictably and that better solution quality is achievable when more execution time is available. There are, however, two pieces of not so good news: (a) the cost of improving solutions beyond a certain point may be prohibitive and the resulting gain vary slight (as evidenced by the flattening out of all curves in Figure 2), and (b) the time budget required before a more powerful heuristic is preferable to a less powerful one grows nonlinearly with the size of the graph when one is limited to our current arsenal of heuristics.

The following issues deserve further study:

- Are there combinations of averaging and insertion that require less than quadratic time per iteration and still significantly outperform averaging?
- Should more computational effort be put into good initial solutions rather than iterative improvement? Guided breadth-first search is not at all sophisticated and requires only linear execution time. Even within linear time we can do more (e.g. partition into biconnected components). Whether this will lead to better results when combined with iterative improvement remains to be seen.
- How universal are these results (and ones that we may explore in future work)? Will they carry over to classes based on denser graphs? Or graphs that are structurally different from those encountered in this study?
- Can we learn anything by carefully observing the behavior of the heuristics on individual graphs in our sample?

What accounts for the differences in t_1 and t_2 among different graphs and different initial solutions? Can we use this information to improve our heuristics?

REFERENCES

- [1] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
- [2] F. T. Leighton. New Lower Bound Techniques for VLSI. *Math. Systems Theory*, 17:47–70, 1984.
- [3] M. Marek-Sadowska and M. Sarrafzadeh. The Crossing Distribution Problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(4):423–433, April 1995.
- [4] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrt’o. On bipartite drawings and the linear arrangement problem. In *Lecture Notes in Computer Science*, number 1467, pages 55–68. Springer Verlag, 1997.
- [5] D. Ghosh, F. Brglez, and M. Stallmann. Hypercrossing Number: A New and Effective Cost Function for Cell Placement Optimization. Technical Report 1998-TR@CBL-12-Ghosh, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, December 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-TR@CBL-12-Ghosh>.
- [6] F. Harary and A. J. Schwenk. A New Crossing Number for Bipartite Graphs. *Utilitas Mathematica*, 1:203–209, 1972.
- [7] M. R. Garey and D. S. Johnson. Crossing Number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4:312–316, 1983.
- [8] N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, 1995.
- [9] M. Stallmann, F. Brglez, and D. Ghosh. Heuristics and Experimental Design for Bigraph Crossing Number Minimization. In *Proceedings of the First Workshop on Algorithm Engineering and Experimentation (ALENEX’99)*. Springer Verlag, Lecture Notes in Computer Science, January 1999. Also available at <http://www.cbl.ncsu.edu/publications/#1999-ALENEX-Stallmann>.
- [10] F. Brglez and R. Drechsler. Design of Experiments in CAD: Context and New Data Sets for ISCAS’99. In *IEEE 1999 International Symposium on Circuits and Systems - ISCAS’99*, May 1999. A reprint also accessible from <http://www.cbl.ncsu.edu/experiments/DoEArchives/1999-ISCAS>.
- [11] D. Ghosh and F. Brglez. Equivalence Classes of Circuit Mutants for Experimental Design. In *IEEE 1999 International Symposium on Circuits and Systems - ISCAS’99*, May 1999. A reprint also accessible from <http://www.cbl.ncsu.edu/experiments/DoEArchives/1999-ISCAS>.