

Heuristics and Experimental Design for Bigraph Crossing Number Minimization

Matthias Stallmann, Franc Brglez*, and Debabrata Ghosh*

Department of Computer Science, PO Box 8206
N.C. State University
Raleigh, NC 27695-8206 USA
<http://www.cbl.ncsu.edu/experiments/>

Abstract. The *bigraph crossing problem*, embedding the two vertex sets of a bipartite graph $G = (V_0, V_1, E)$ along two parallel lines so that edge crossings are minimized, has application to circuit layout and graph drawing. We consider the case where both V_0 and V_1 can be permuted arbitrarily — both this and the case where the order of one vertex set is fixed are NP-hard. Two new heuristics that perform well on sparse graphs such as occur in circuit layout problems are presented. The new heuristics outperform existing heuristics on graph classes that range from application-specific to random. Our experimental design methodology ensures that differences in performance are statistically significant and not the result of minor variations in graph structure or input order.

Keywords. crossing number minimization in graphs, graph equivalence classes, design of experiments

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 [4, 7, 10, 23, 29], (2) reducing the wiring congestion and crosstalk in VLSI circuits, which in turn may reduce the total wire length and the layout area [19, 22, 25, 26]. This paper is about *bigraph crossing* defined as follows for any bipartite graph (bigraph) $G = (V_0, V_1, E)$ [16]: Let G be embedded in the plane so that the nodes in V_i occupy distinct positions on the line $y = i$ and the edges

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

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 [11] proved that it is NP-hard to compute $C(G)$. Detection of a *bipolar* graph, a bigraph G for which $C(G) = 0$, however, is easy [15].

Previous work on bigraph crossing has focused primarily on the *fixed-layer* (previous of the) problem, namely computing $C(G)$ subject to the constraint that the permutation of V_0 on $y = 0$ must stay fixed. Even that is NP-hard [9]. Work on heuristics for both fixed-layer and general bigraph crossing has mostly been theoretical [4, 8, 9, 20, 21]. Experimental evaluation has focused on dense graphs, for which good lower bounds are available [5, 18], and sparse random graphs [18]. Graphs arising in circuit design are very sparse and highly structured. Moreover, the `dot` package [10], used widely for graph drawing, has not been compared with other heuristics. This paper addresses these shortcomings by (a) evaluating heuristics based on their performance on general (rather than fixed-layer) bigraph crossing, (b) presenting new heuristics that perform well on instances related to circuit design and random graphs of the same sparsity, (c) doing careful comparison of new heuristics with `dot` and other heuristics in the literature, and (d) presenting an experimental methodology that allows us to validate heuristics on data that ranges from application-specific to random.

The performance of a heuristic can vary widely even on a single input graph, depending on the order in which the input is presented. This motivates us to define a *presentation* (of a graph G) as $\langle G, \pi_0, \pi_1 \rangle$, where π_i is a permutation of V_i . As any heuristic implicitly sequences the input when it reads data, the presentation captures essential information about any of the common ways of describing a bigraph. If described as a list of neighbors for each V_0 node, π_0 describes the order of appearance of the V_0 nodes while π_1 is used to sort the adjacency lists. A list of edges is sorted using π_0 as primary key and π_1 as secondary key. Quite conveniently, a presentation also yields an embedding of G : use π_i to sequence the V_i vertices along $y = i$. Let $C(\langle G, \pi_0, \pi_1 \rangle)$ denote its crossing number. The object of the bigraph crossing problem, then, is to compute (or approximate) $C(G) = \min_{\pi_0, \pi_1} C(\langle G, \pi_0, \pi_1 \rangle)$.

Pushing this idea further, a heuristic h is a mapping from one graph presentation to another, that is $h(\langle G, \pi_0, \pi_1 \rangle) = \langle G, \pi'_0, \pi'_1 \rangle$ and we can study its behavior statistically by looking at how a distribution on random $\langle G, \pi_0, \pi_1 \rangle$ drawn from a class of presentations imposes a distribution on $C(h(\langle G, \pi_0, \pi_1 \rangle))$. Recently the distribution of $C(h(\langle G', \pi_0, \pi_1 \rangle))$ using a specific heuristic h became an important factor in characterizing a graph [14]: π_0 and π_1 were chosen randomly and G' was chosen from a well-defined set of perturbations of G . In this paper, we demonstrate the converse: sets of graph presentations have an important role in the design of experiments such that the performance of the various known bigraph crossing heuristics can be measured with statistical significance. Moreover, the experiments have stimulated the development of a new heuris-

tics and we demonstrate that the improved performance is due to significant improvements of the algorithms and not due to chance.

The paper is organized as follows: Section 2 motivates the proposed approach, Section 3 presents the proposed heuristics, both existing and new, Section 4 describes the design of our experiments, in terms of heuristics applied and data sets used, Section 5 summarizes experimental results, and Section 6 presents tentative conclusions and suggests further work. An appendix describes how to access and use our data and programs via the world-wide web.

2 Background and Motivation

Routinely, we use `dot` [10] as the crossing number minimization algorithm to characterize and distinguish equivalence classes of graph presentations as defined in [14]. In 1997, we posted on the Web results of a very basic experiment on three families of equivalence classes of bigraph presentations: (a) *isomorphism* classes (classes in which G remains fixed but π_0 and π_1 vary) based on a biplanar graph, (b) isomorphism classes based on a graph with two cycles joined at an articulation point ($C(G)$ is known for this graph, which we call *cyclic*), (c) perturbed classes (presentations have the form $\langle G', \pi_0, \pi_1 \rangle$ where G' is a well-defined perturbation of G) based on the graphs in (b) [2]. It quickly became apparent that `dot` performs far from optimal, even for graphs with as few as 9 nodes.

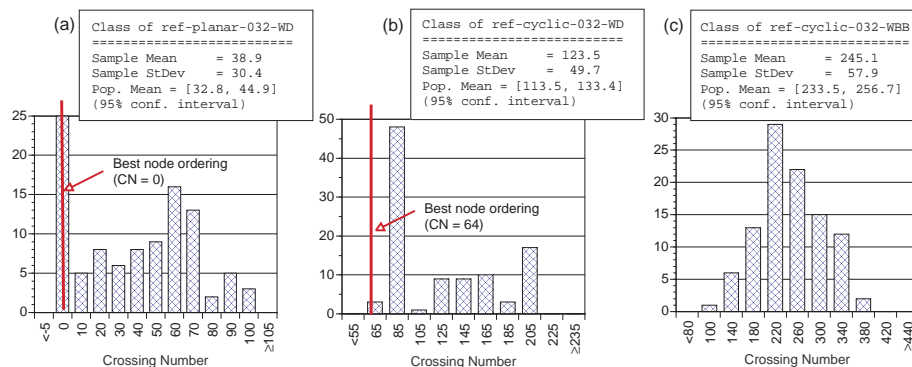


Fig. 1. Crossing number distributions and statistics observed with program `dot` on three equivalence classes of bigraphs: (a) 100 instances of isomorphic biplanar graphs, (b) 100 instances of isomorphic non-planar graphs with a minimum crossing number of 64, (c) 100 instances of mutant graphs of (b).

An example of the observed performance with `dot` on the three classes is shown in Figure 1. All graphs analyzed in Figure 1 are of comparable size and density: the biplanar graph has 128 nodes and 129 edges, the cyclic graph has

131 nodes and 132 edges and a crossing number of 64, each of the mutant graphs has 131 nodes and 132 edges and the crossing number is expected to be a random variable due to the construction of the mutant graphs. The histograms in Figure 1 demonstrate that for graphs of this size, $C(\text{dot}(\langle G, \pi_0, \pi_1 \rangle))$, the crossing number as reported by `dot`, behaves as a random variable not only for the perturbed class in Figure 1(c) as expected, but also for the two isomorphism classes where, ideally, each distribution should have variance of 0 and population means of 0 for Figure 1(a) and 64 for Figure 1(b). Notably, only 25 of 100 graph presentations in Figure 1(a) achieve $C(\text{dot}(\langle G, \pi_0, \pi_1 \rangle)) = 0$; there are 10 presentations whose crossing number distribution ranges from 80 to 100. For the 100 graph presentations in Figure 1(b), the best reported crossing number is 70, and there are 5 presentations with $C(\text{dot}(\langle G, \pi_0, \pi_1 \rangle)) = 211$!

The results of simple experiments summarized in Figure 1 provide a strong motivation for the premise of this paper: *by reducing the dependency of the crossing number minimization algorithm on the initial order of nodes in the graph, we will have improved the overall performance of the algorithm.*

We illustrate the nature of the problem with the example in Figure 2. The

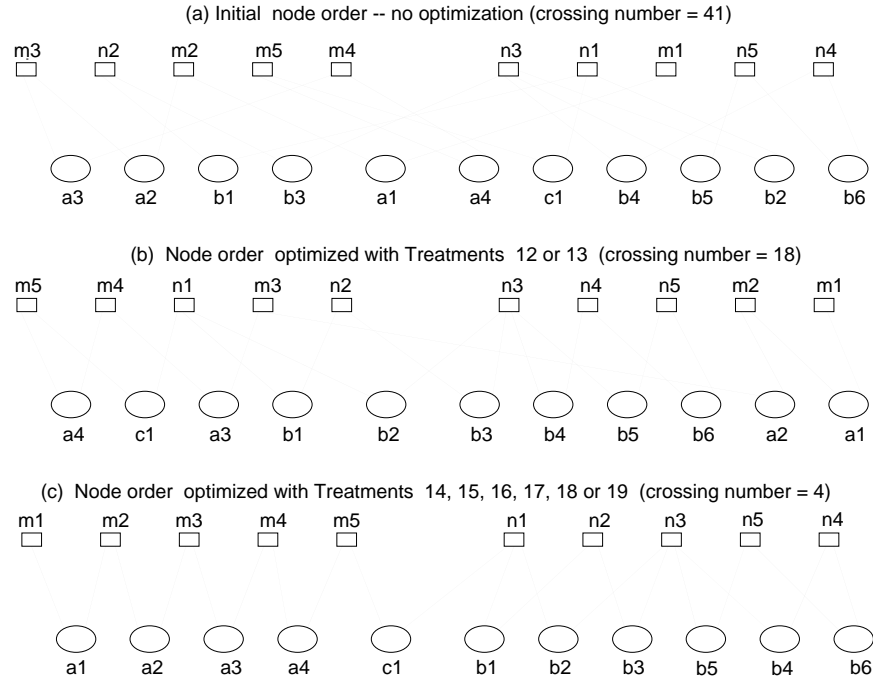


Fig. 2. A simple graph example illustrating: (a) a presentation that is found ‘difficult’ for the `dot` heuristic, (b) the presentation found by `dot`, (c) an optimum presentation found by heuristic combinations `tr14-tr19` reported in this paper.

initial order of the bigraph in Figure 2(a) reports a crossing number of 41. The best node ordering achieved by `dot` reports a crossing number of 18 and is shown in Figure 2(b). However, the optimal node ordering achieved by several heuristics reported in this paper results in a crossing number of 4 and is shown in Figure 2(c).

`Dot`'s initial ordering, based on breadth-first search, folds the path from `m1` to `n1` into an awkward position so that even arbitrarily many iterations of the subsequent median heuristic (`dot` stops at 24 iterations) cannot disentangle it. The initial ordering imposed by our two-pass *guided breadth-first search*, described in the next section, is shown in Figure 2(c). It is optimal.

3 Overview of the Heuristics

Our heuristics (experimental treatments) follow the scheme outlined in the classical paper by Warfield [29]. There are two phases: an initial ordering and iterative improvement. The former involves some global computation on the graph to sequence the nodes in each layer and the latter repeatedly solves a fixed-layer problem on alternating sides. Various combinations of initial ordering and iterative improvement make up the experimental treatments discussed in Section 4.

3.1 Initial Ordering

Two observations suggest that a carefully computed initial ordering can avoid traps for subsequent attempts at improvement without incurring prohibitive execution time. First, a connected graph is biplanar iff it is a *comb* (tree that becomes a path when its leaves are removed) [15]. The embedding of a comb can be computed easily in linear time [6]: the trick is to embed the *spine* (path that remains when leaves are deleted from the comb) from left to right, zig-zagging from layer to layer, and inserting any leaves attached to a spine node to the left of the next spine node. Second, a cycle of length $2n$ has an optimum embedding with $n - 1$ crossings [16] and that embedding is achieved if the nodes are sequenced in breadth-first order starting at any node of the cycle.

We use two initial orderings in our experiments in addition to the random ordering (obtained by using the input presentation), for a total of 3 possibilities. One ordering uses a breadth-first search starting at a random node. This always gets the optimal solution for a simple cycle, and tends to perform better than input ordering on the classes of graph presentations we looked at. The `dot` heuristic also starts with a breadth-first search, but has additional features related to aesthetic objectives other than minimizing crossing.

Our contribution is an initial ordering heuristic called *guided breadth-first search (GBFS)*. Two passes of breadth-first search are done. In the first pass, each node v is given $dist[v]$, the distance from the start node, and $depth[v]$, the maximum value of $dist[w]$ achieved by any descendant w of v in the breadth-first search tree. The second pass begins the search at a node s for which $dist[s]$ is maximized. Adjacency lists are sorted by increasing *depth* and ties are broken

by visiting nodes w with larger $dist[w]$ first. Sequence numbers are assigned to nodes based on the order of visitation in the second search. Finally, the sequence numbers are used to sort nodes on each side of the bigraph. All of the above can be accomplished easily in linear time: the $depth$ values can be calculated as nodes are visited in reverse order at the end of the first bfs ($depth[v] = \max_{w \text{ a child of } v} depth[w]$), and all sorting is based on values that range from 1 to n (bin sorts of all adjacency lists can be combined). Our actual implementation is not linear time — it uses insertion sort — but it runs fast enough in practice (and timing is not an issue we address directly in this paper).

Figure 3 shows the two breadth-first searches of GBFS on the example in Figure 2. The first search begins at $n4$, but that choice is completely arbitrary.

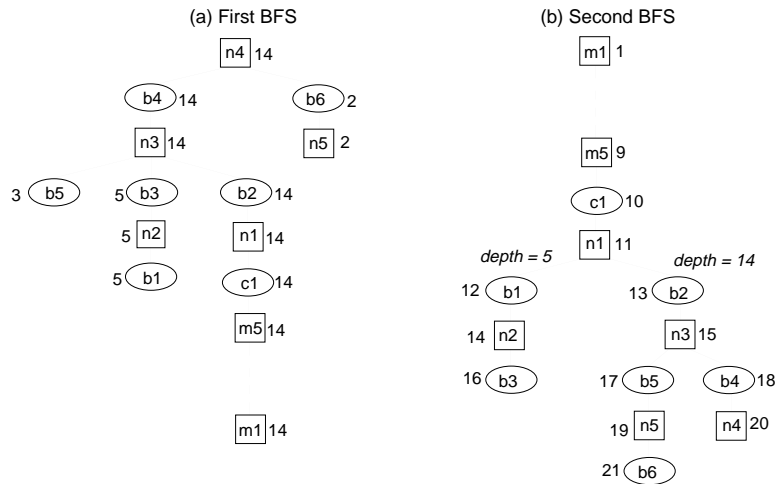


Fig. 3. GBFS starting at node $n4$ of Figure 2(a). Numbers next to nodes indicate depth in (a) and sequence numbers in (b).

In some cases starting the first bfs at a node of maximum degree improves performance of GBFS considerably. A max-degree node is likely to be an articulation point included in more than one cycle, and beginning the search there reduces the likelihood that two cycles that can be embedded without interference will cross each other. We include this enhancement in our implementation, though it appears to be superfluous when a good iterative improvement strategy follows GBFS. It is easy to show that GBFS always obtains optimal solutions for biplanar graphs and simple cycles.

3.2 Iterative Improvement

To improve upon the initial orderings discussed above, we repeatedly apply a heuristic for fixed layer crossing. Our experiments use two popular previously-

known heuristics and a new heuristic called *adaptive insertion*. These heuristics, described in detail below, are used separately or in combination. Regardless of which heuristic is used, our implementation has a parameter that governs the maximum number of iterations without improvement, set arbitrarily at 24. After each application of the heuristic, the current ordering is saved if it has fewer crossings than any ordering observed so far. The discovery of a better solution also resets the iteration count to 0. For purposes of the discussion below, we define each heuristic to mean all iterations of it. An *iteration* consists of two passes, one on each layer, where a *pass* is a single application of the original heuristic for the fixed layer problem.

The *median* heuristic treats the neighbors of each node as a set of integers representing their ordinal numbers on the opposite side. Nodes are sorted using the medians of these sets as keys. Implementations of the median heuristic differ in how the median of a set of even cardinality is computed. Ordinarily, the median would be defined as the mean of the two middle elements. The `dot` package [10] uses the mean when there are exactly two elements and a biased mean (biased toward the side on which neighbors are closer together in the ordering) otherwise. The median heuristic for which theoretical bounds are known (see [4, 9, 20]) always uses the smaller of the two candidates, but with the added condition that, in case of ties, nodes with odd degree always precede those of even degree. We adopt this latter median heuristic in our experiments. Assuming random initial ordering and a stable sort, the probability that the median heuristic embeds a simple path optimally is $O(1/n)$. However, it almost always embeds a simple cycle optimally (without the limit of 24 iterations, it would succeed every time).

The *barycenter* heuristic uses the mean of the set of neighboring positions, rather than the median, as a key for sorting (the name comes from the fact that it's a one-dimensional analog of Tutte's barycenter method for drawing graphs [27, 28]). It performs poorly on paths, cycles, and other very sparse highly-structured graphs (since these require decisive movement of degree-2 nodes). On graphs that are more random and/or have several nodes of high degree, barycenter does better than median (high-degree nodes need to be centrally located wrt to their neighbors).

Sometimes a mix of median and barycenter does better than either. Our implementation has a parameter α and sorts nodes using keys $\alpha B + (1 - \alpha)M$, where B and M are the barycenter and median keys, respectively. When the mix is a significant improvement, the best choice of α appears to be 0.5 (as opposed to 0.25 or 0.75).

Each pass of the median heuristic can be implemented in linear time (the keys used for sorting are integers in the range $0, \dots, n - 1$), while the barycenter or mixed heuristics require $O(m + n \log n)$ per pass (m is the number of edges, n the number of nodes).

The new ordering computed by a single pass of the median or barycenter heuristic is independent of any cost considerations and therefore non-adaptive in the sense that any rearrangement is done whether or not it decreases cost. The

`dot` heuristic, however, has a final phase that decides whether or not to exchange the nodes (at positions) i and $i + 1$ on layer ℓ based on whether $d_\ell(i, i + 1)$, the resulting difference in crossing number, is negative (represents improvement). The value $d_\ell(i, i + 1)$ depends only on the relative positions of neighbors of nodes i and $i + 1$ on layer $1 - \ell$ and is therefore easy to compute [4, pp. 281–283].

Adaptive insertion, our contribution to iterative improvement, generalizes the conditional exchange idea and considers the effect of inserting node i before (after) any node $j < i$ ($j > i$) on layer ℓ . When $j < i$ the resulting cost change, $D_\ell(i, j)$, is $\sum_{j \leq k < i} d_\ell(i, k)$ — the effect of the insertion is that of a series of swaps of i with $i - 1, \dots, j$ (situation is symmetric when $j > i$).

One pass of adaptive insertion does a right-to-left sweep of nodes on a layer ℓ . If the current node i has not already been inserted, the position j , before or after, with the best $D_\ell(i, j)$ is found (nodes are not allowed to stay in place even if any insertion would increase the number of crossings).

Consider the example in Figure 2(a) and suppose adaptive insertion is applied to the upper layer $\ell = 1$. The rightmost node is **n4** and $D_1(\mathbf{n4}, \mathbf{n5}) = d_1(\mathbf{n4}, \mathbf{n5}) = -1$. Since $d_1(\mathbf{n4}, \mathbf{m1}) = 2$, the value of $D_1(\mathbf{n4}, \mathbf{m1})$ increases to 1. As we move farther to the left, considering insertion positions for **n4**, the value of $D_1(\mathbf{n4}, x)$ continues to increase, so the best choice is to insert **n4** before **n5**. The sweep then moves left to **n5**, which is skipped, having already been inserted (**n4** does not have an opportunity to be inserted in this phase; giving it one would be counterproductive, since it would simply be swapped with **n5**). Next **m1** finds its optimal position before **m5** with $D_1(\mathbf{m1}, \mathbf{m5}) = -5$. And so on. Figure 4 shows the results of the completed pass of adaptive insertion on the graph of Figure 2(a). The node **m3** was forced to swap with **m2** (the least of evils) even though this caused a net gain of 3 crossings.

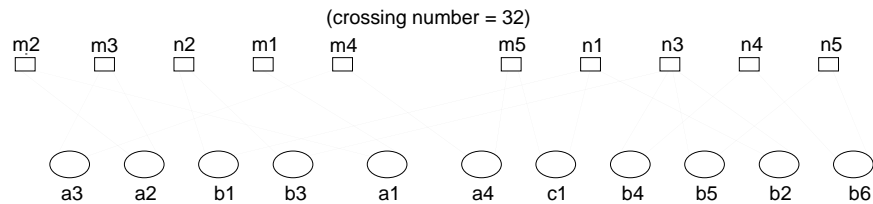


Fig. 4. One pass of adaptive insertion starting with presentation shown in Figure 2(a).

Adaptive insertion seems to do well where the median does badly and vice-versa. An effective combination alternates iterations of adaptive insertion with iterations of the median/barycenter mix. This, combined with an initial GBFS ordering, is the best overall performer among our heuristics.

One pass of adaptive insertion takes $O(m^2)$ time. If all adjacency lists are initially sorted using layer $1 - \ell$ positions as keys (this can be done in linear time), the calculation of $d_\ell(i, k)$, and hence $D_\ell(i, k)$, for all $k \neq i$ takes time

$O(mk)$ where $k = \text{deg}(i)$. Summing over all nodes i gives the $O(m^2)$ bound. For denser graphs the time per node can be reduced to $O(m \log k)$ using a simple data structure, for an overall bound of $O(mn \log(m/n))$ per pass.

4 Experimental Design

Experiments with results such as summarized in Figure 1 demonstrate that particular executions of a heuristic to solve the optimum node ordering in bigraphs offer no indication of the quality of the solutions produced in general. Changing the starting point for the problem instance can induce unpredictable variability of results when experiments are repeated.

Two of the fundamental principles of experimental design are *randomization* and *replication*. We adopt these principles for the experimental evaluation of heuristics by (1) creating a *presentation equivalence class*, (2) repeating the experiments for each member in the class, and (3) making the equivalence class and the heuristics available so that other researchers can repeat our experiments (or variations of them; see the Appendix for more details). The basic abstractions for such experiments include [1]:

1. an equivalence class of experimental subjects, eligible for a treatment;
2. application of a specific treatment;
3. statistical evaluation of treatment effectiveness.

Here, a *treatment* is synonymous with a *heuristic* and an equivalence class of experimental subjects is synonymous with a graph presentation class. Figure 5 illustrates these abstractions in a generic flow. The cost index, minimized by permuting the nodes at both levels of the graph, is $C(h(\langle G, \pi_0, \pi_1 \rangle))$, where h is any of the treatments in Figure 5.

The treatments can be divided into three main groups based on the initial ordering (see the previous section) used: **tr00-tr05** represent random (input) order, with **tr00** playing the special role of a placebo treatment; **tr06-tr11** represent breadth-first ordering; and **tr14-tr19** represent ordering obtained from our GBFS. The remaining treatments, **tr12** and **tr13** are dot with 24 and 48 iterations, respectively.

The iterative improvement heuristic used (previous section) determines the treatment order within each group: the first treatment (**tr00**, **tr06**, **tr14**) does no iterative improvement; the second through fourth (**tr01-tr03**, **tr07-tr09**, **tr15-tr17**) do the median, median/barycenter mix, and barycenter; and the last two (**tr04-tr05**, **tr10-tr11**, **tr18-tr19**) do adaptive insertion, either by itself or alternated with a median/barycenter mix.

Experimental subjects are drawn from circuit layout problems. The example in Figure 2, is representative of graphs that can be extracted from VLSI designs such as shown in Figure 6. This particular example is based on the largest connected component found in a dag, referenced as a **classifier controller** or **cc** in [24]. Since the circuit **cc** implements the controller, there are a few nodes with a large out-degree (or fanout) that introduce unavoidable edge crossings; note however, that there is also a small biplanar region. Before introducing

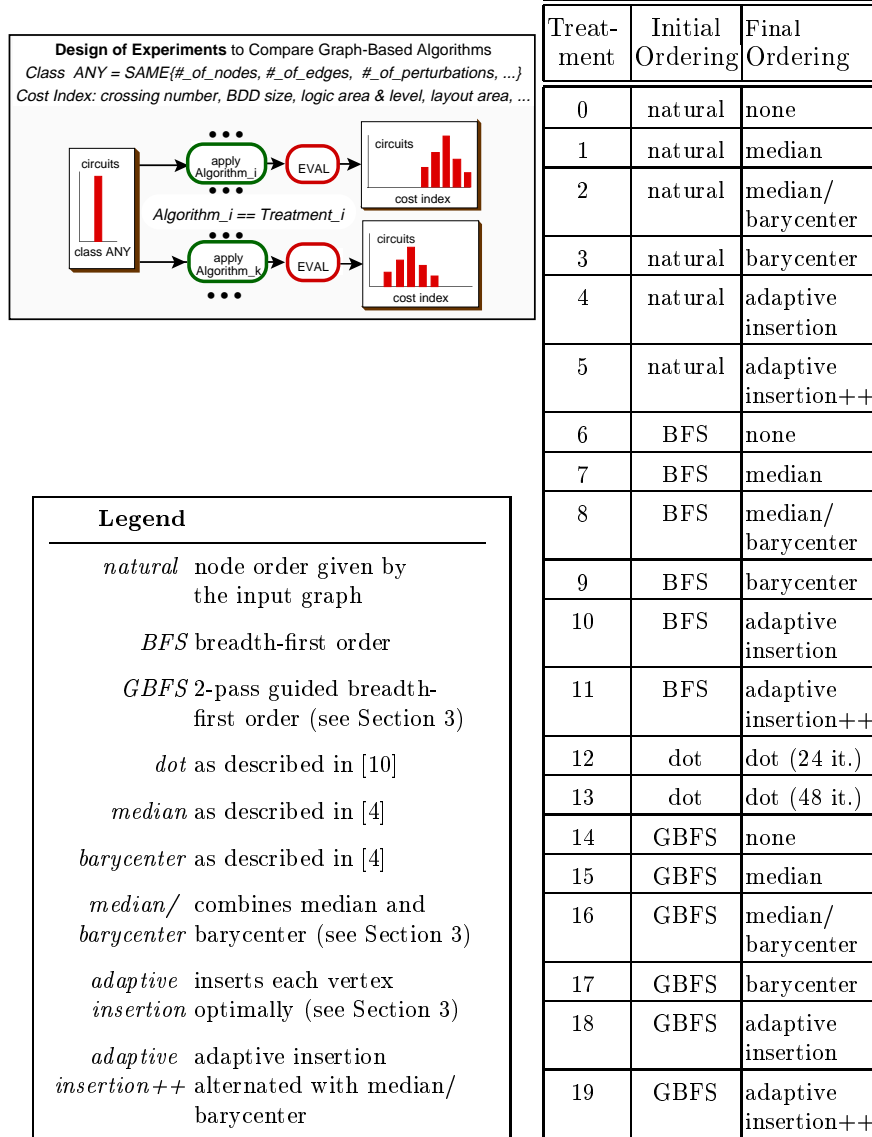


Fig. 5. Design of experiments with bigraph equivalence classes to compare the crossing numbers under distinctive vertex ordering algorithms (treatments).

families of parameterized graphs that have properties of the graph in Figure 7, we briefly describe the graph mutation process as introduced in [14].

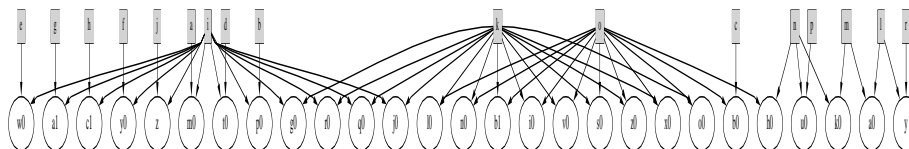


Fig. 6. Example of a bigraph, extracted as a *connected component* from a dag of a VLSI circuit.

The graphs in Figures 2 and 6 are directed bigraphs: the square nodes represent signal source nodes or *net nodes*, the oval nodes represent the signal processing nodes or *cell nodes*. Since the graph under consideration has two layers only, there is no need to define sink nodes. In VLSI circuits, the out-degree or *fanout* of net nodes is assumed to be a loosely bounded random variable, the in-degree or *fan-in* of the cell nodes is assumed to be a tightly bounded random variable. By design, and for simplicity of inducing the equivalence classes of bigraph presentations, all cell nodes considered in this paper have a fan-in of 2. The graph such as shown in Figure 2 illustrates an instance of a *reference graph* G_r with a *characteristic signature* $\sigma = \{10, 11\}$ where 10 is the total number of net nodes and 11 is the total number of 2-input cell nodes.

From each reference graph G_r , we can define three equivalence classes of *derived presentations* (see [14] for more details and more possibilities): (1) the *isomorphism class* for G_r (designated by the suffix -WD in the figures reporting results), consisting of presentations $\langle G_r, \pi_0, \pi_1 \rangle$ (only the permutation of nodes on each layer differs). (2) the *mutant class* for G_r (suffix -WBB), consisting of presentations $\langle G'_r, \pi_0, \pi_1 \rangle$, where G'_r is a random connected graph with the same signature as G_r (we assume G_r is connected as well), that is, the same number of degree-2 cell nodes on one layer (hence the same number of edges as well), and the same number of net nodes, and (3) the *random class* for G_r (suffix -WRD), consisting of presentations $\langle G'_r, \pi_0, \pi_1 \rangle$, where G'_r is completely random with the same number of nodes on each layer and the same number of edges as G_r (G'_r is not necessarily connected, nor does it have fan-in 2 for cell nodes, but it has no isolated nodes). We took a special precaution to assign to each instance of any class the following properties:

- P1:** the order of all cell and net nodes in each presentation is random relative to all other members in the class (that is, π_0 and π_1 are uniformly random),
- P2:** the names of all cell and net nodes in each presentation are assigned randomly relative to all other members in the class.

Properties **P1** and **P2** are essential to good experimental design and must be maintained universally for all equivalence classes. The purpose of **P1** is clear.

Without **P2**, some programs that rely on hashing the input data may *unknowingly undo* the randomization of input presentations and confound the experiments. An important lesson on this subject has been learned and reported in [17].

In addition to using the `cc` graph directly, we use three types of reference graphs in sizes that are increasing powers of 2: (1) the *biplanar* type of size q is a comb with $2q + 1$ net nodes and $2q$ cell nodes for a total of $4q + 1$ nodes and $4q$ edges, (2) the *cyclic* type of size q consists of two simple cycles each having $q + 2$ nodes and edges and joined at a net node (which becomes an articulation point), and (3) the *combined* type of size q is a biplanar graph of size q joined to a cyclic graph of size q via an additional connector (cell) node for a total of $8q + 5$ nodes and $8q + 6$ edges. Figure 2 is an example of a combined graph of size 2.

In the following section we report the results of 12 distinct experiments, consisting of isomorphism classes, mutant classes, and random classes based on the `cc` graph and each of the reference graph types biplanar, cyclic, and combined. The increasing sizes indicate asymptotic trends in the relative behavior of the heuristics.

5 Experimental Results

The experiments are based on the model shown in Figure 5. Treatments 0–19 are applied to *all* classes of data described in the paper. For each presentation of input $\langle G, \pi_0, \pi_1 \rangle$, a treatment h_k induces a treated representation $h_k(\langle G, \pi_0, \pi_1 \rangle)$ as $\langle G, \pi'_0, \pi'_1 \rangle$ which is in turn evaluated by a *common crossing number evaluator*, `cn_eval`. The evaluator (1) reads a file of the graph description in `dot` format, as well as a companion file that simply stores the vertices V_i in the order determined by π'_i , and (2) reports the crossing number of the presentation.

All classes of input data are organized for easy access through the Web, like the variety of illustrative experimental designs reported earlier [3]. As in [3], web-posting includes complete tables of crossing number results for all treatments and all classes of input data, along with statistical summaries that include confidence intervals of the reported means, and t-tests to analyze the significance of reported differences between the means. Since the treatment 19 is clearly the best overall treatment reported in the current work, we also post node permutations π'_0, π'_1 along with the respective crossing numbers reported by this treatment. Readers of this text should look under http://www.cb1.ncsu.edu/experiments/-DoE_Archives/DoE_0003 for completed archives of experimental data and programs presented in this paper (see the Appendix for more information).

Due to space limitations, we confine the statistical summaries of our experiments to illustrations shown in Figure 7 and 8.

5.1 Figure 7

Figure 7 shows treatments sorted by mean cost on the `cc` reference graph. The best heuristic overall is a combination of several: GBFS initial order, followed

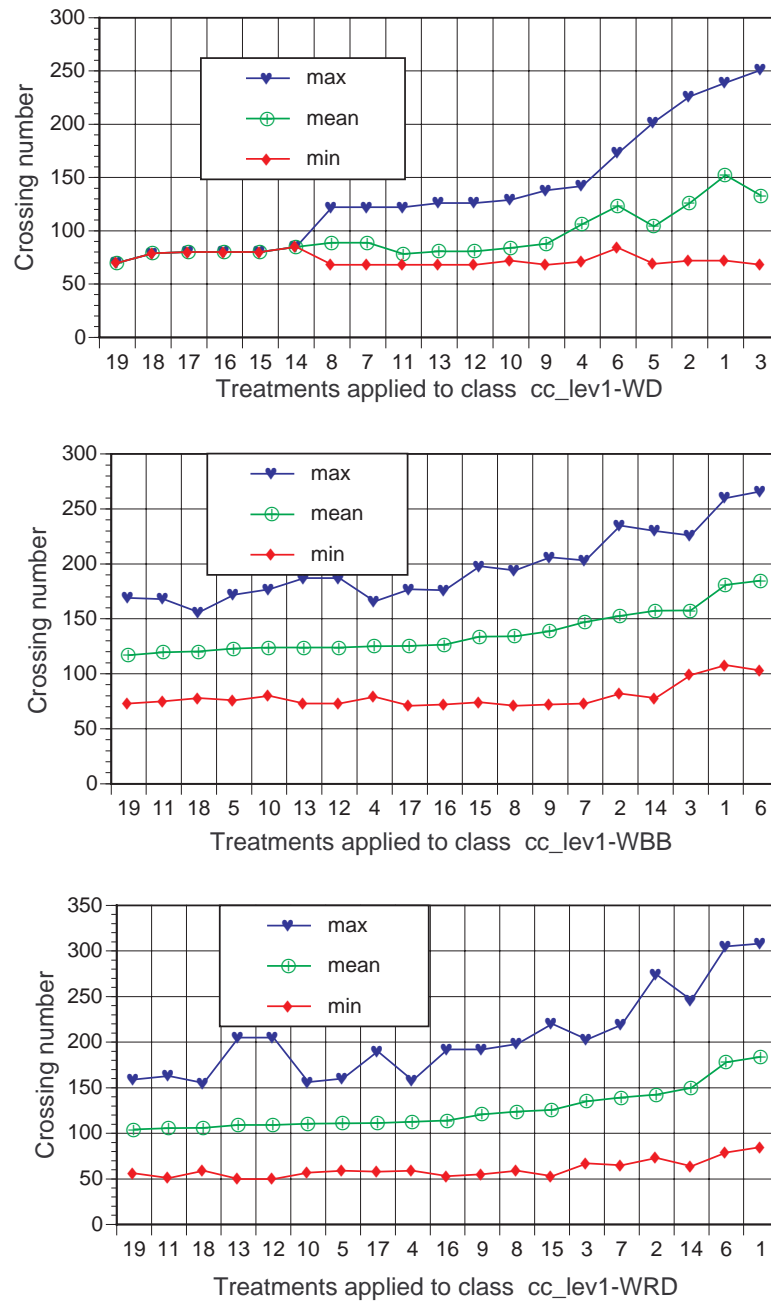


Fig. 7. Crossing number reports for treatments 0-19 applied to three equivalence classes derived from the reference bigraph `cc_lev1`.

by adaptive insertion alternated with a median/barycenter mix. This multi-heuristic combination consistently does better on all presentation classes and also has better asymptotic behavior with respect to solution cost than any of the others. The placebo, `tr00`, is not shown: random solutions are far worse than those produced by any of the heuristics. For the isomorphic class, they range from 440 to 752 crossings; for the mutant class from 501 to 780; and for the random class from 314 to 818.

Even though the difference between bfs and GBFS is dramatic for the isomorphism class when these heuristics act alone (`tr06` and `tr14`), the difference decreases significantly as iterative improvement is added (`tr10` vs. `tr18`, `tr11` vs. `tr19`) or the classes become more random (mutant and random versus isomorphism). This is not surprising since GBFS was inspired by the circuit-derived examples.

Adaptive insertion is clearly a better strategy for iterative improvement than any of the others, but it is also the most time intensive. This suggests that bigraph crossing heuristics have not yet reached a point of diminishing return in terms of investment in execution time. Whether this should be exploited by increasing the number of iterations without improvement or by developing more sophisticated local search strategies remains unclear, most likely the latter.

5.2 Figure 8

Here we summarize results of the remaining 9 experiments (3 graph types, each with 3 different presentation classes). The plots that show only `tr19`, the best overall heuristic, illustrate the growth in objective function value on a log/log scale. Except for the biplanar isomorphism class, the number of crossings reported grows polynomially with the number of edges, the slope representing the exponent. For the biplanar classes, the difficulty (from the point of view of `tr19`) increases with the degree of randomness. The mutant class (`wbb`) contains trees that are not biplanar, but no cycles, while the random class may also contain cycles. This suggests that non-biplanar trees can be pretty difficult, but not as difficult as more general bigraphs. In the cyclic and combined (`multi1`) classes, the mutants are more difficult than the random bigraphs: recall that the mutants have a single connected component while the random graphs, because of their sparsity, are unlikely to have even a large connected component relative to the rest of the graph.

The other plots show that the difference between crossing numbers obtained by `tr19` and three competitors increases with increasing bigraph size. A log/linear scale is used here to make the separation more visible, but it should be pointed out that all of the differences grow nonlinearly (they curve upward even on a linear/linear scale but the lines are no longer as distinct from each other). `Dot` (represented by `tr13`) is clearly a competitive heuristic and is put at a disadvantage with respect to the adaptive insertion heuristics as the graph size increases because of its fixed iteration limit. An adaptive bound on the number of iterations, as the authors of [10] suggest might improve its relative performance.

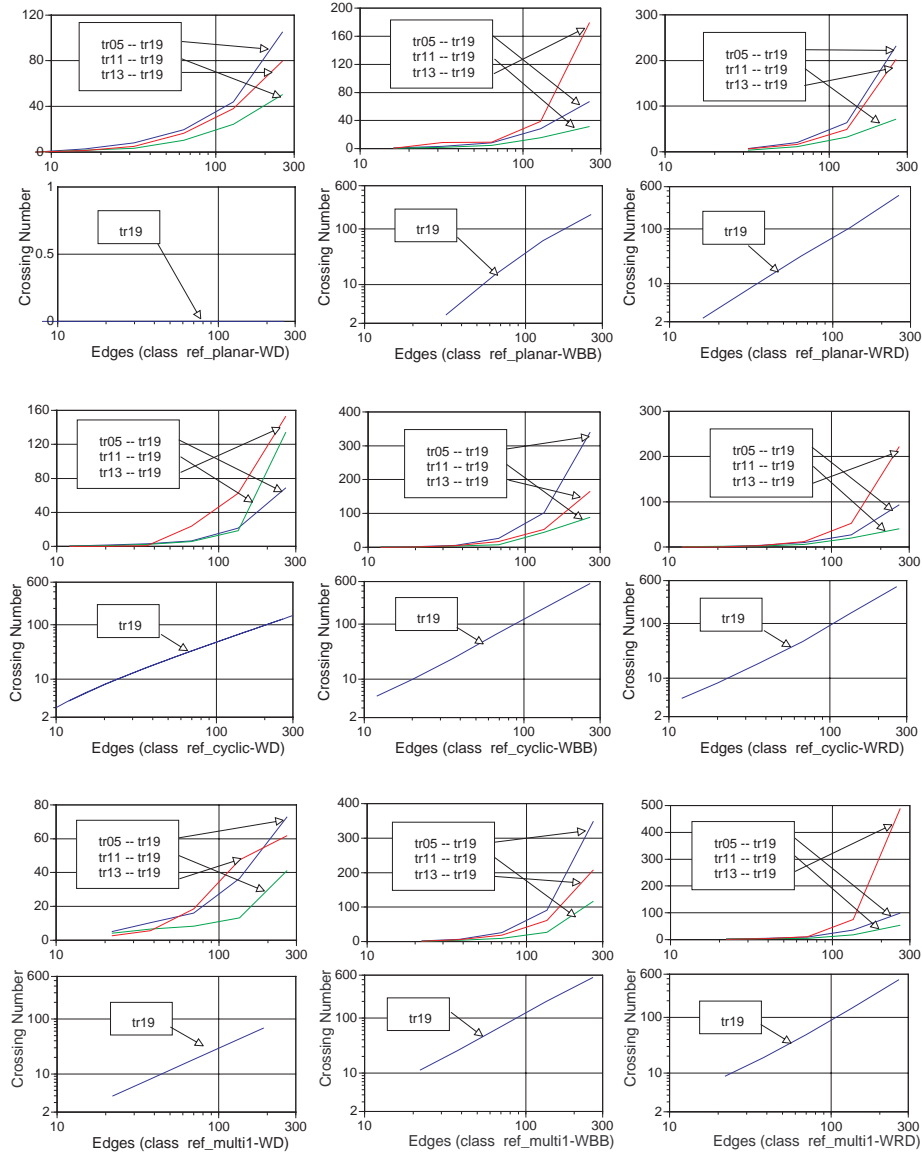


Fig. 8. Summary of asymptotic experiments on minimizing the *crossing number mean* with four treatments (tr05, tr011, tr13, tr19), applied to three parameterized families of reference bigraphs (biplanar, cyclic, and combined, i.e. multi1), each bigraph inducing three equivalence graph classes, each class with the same number of edges and nodes (and no isolated nodes): isomorphism class (WD), signature-invariant and component-invariant mutant class (WBB), and random class (WRD).

Our hypothesis that better initial orderings make a difference is clearly supported, even in the presence of sophisticated iterative improvement, as seen by the growing gaps between `tr19` and `tr11`, and again between `tr11` and `tr05`; all three use `adaptive-insertion++` for iterative improvement.

6 Conclusions and Further Work

This study is only the beginning of what we hope is a new approach to experimental study of bigraph crossing and other intractable problems. Input data and treated output can be shared and verified so that different groups working on the same problem can conduct repeatable experiments. Better heuristics are often developed through a detailed understanding of why specific instances present difficulties, and such understanding is made more likely when data is a significant component of the experimental design.

Some directions to pursue later include

- Better lower bounds for sparse graphs: these could be based on the fact that a tree requires at least as many crossings as the minimum number of edges that need to be deleted in order to produce a comb and on the theoretical lower bound for cycles (it appears that these can be combined additively for a spanning tree of a graph and its fundamental cycles).
- Even better initial ordering: there are simple examples of trees on which GBFS does poorly. A useful approach might be to find biconnected components and then try to optimize the underlying tree structure. For a tree T , it is known that $C(T)$ can be computed in polynomial time [25], although the algorithm appears to be complicated.
- More powerful iterative improvement heuristics: all the ones presented here have at most quadratic time per pass. Are there heuristics that obtain superior solution quality for the price of more computation time?
- Use heuristics as a filter for creating presentation classes in which the distribution of π_0, π_1 changes from uniformly random to some other distribution in order to study which treatments work best in sequence.
- Theoretical work on the median and barycenter heuristics: previous work (as reported in [4]) only addresses the fixed-layer problem. It also appears that the median and barycenter heuristics have interesting convergence properties (they converge quickly to a “local optimum”).
- Relationship between bigraph crossing and other objective functions for layout of VLSI circuits: preliminary experiments [12, 13] indicate a high correlation with wire length in the final routing obtained by at least two different design automation tools. In fact, our bigraph crossing heuristics appear to achieve better wire length than layout heuristics that are specifically designed to minimize wire length. Relevant theoretical work relates bigraph crossing to optimum linear arrangement [25].

ACKNOWLEDGMENTS. Some of the initial crossing number experiments with 2-layer graphs posted on the Web in 1997 were organized and executed by Nevin Kapur. He also contributed the software utilities that support gathering and tabulating data about the experiments and the statistical summaries. Hemang Lavana contributed utilities to dynamically generate web-pages of directories that link the data related to the reported experiments. We appreciate their support.

References

1. F. BRGLEZ, *Design of Experiments to Evaluate CAD Algorithms: Which Improvements Are Due to Improved Heuristic and Which Are Merely Due to Chance?*, Tech. Report 1998-TR@CBL-04-Brglez, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, April 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-TR@CBL-04-Brglez>.
2. F. BRGLEZ, N. KAPUR, AND D. GHOSH, *A CBL PosterNote: Wire Crossing Problem and Benchmarking*, aug 1997. Available from <http://www.cbl.ncsu.edu/DiscussionGroups/Benchmark-reviews/frm00002.html>.
3. F. BRGLEZ (EDITOR), *A Brief Tour From The Home Page on WWW Statistical Experiment Archives: Benchmark Descriptions and Posted Solutions to NP-hard Problems*, Tech. Report 1998-TR@CBL-01-Brglez, Version 1.0, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, February 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-TR@CBL-01-Brglez>.
4. G. DI BATTISTA, P. EADES, R. TAMASSIA, AND I. G. TOLLIS, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.
5. P. EADES AND D. KELLY, *Heuristics for reducing crossings in 2-layered networks*, *Ars Combinatoria*, 21-A (1986), pp. 89–98.
6. P. EADES, B. D. MCKAY, AND N. C. WORMALD, *On an Edge Crossing Problem*, tech. report, Department of Computer Science, University of Newcastle, New South Wales 2308, Australia, 1976.
7. P. EADES AND K. SUGIYAMA, *How to Draw a Directed Graph*, *Journal of Information Processing*, 13 (1990), pp. 424–437.
8. P. EADES AND S. WHITESIDE, *Drawing graphs in two layers*, *Theoretical Computer Science*, 131 (1994), pp. 361–374.
9. P. EADES AND N. C. WORMALD, *Edge Crossings in Drawings of Bipartite Graphs*, *Algorithmica*, 11 (1994), pp. 379–403.
10. E.R. GANSNER, E. KOUTSIFIOS, S.C. NORTH AND K.P. VO, *A Technique for Drawing Directed Graphs*, *IEEE Trans. Software Engg.*, 19 (1993), pp. 214–230. The drawing package `dot` is available from <http://www.research.att.com/sw/tools/graphviz/>.
11. M. R. GAREY AND D. S. JOHNSON, *Crossing Number is NP-complete*, *SIAM J. Algebraic Discrete Methods*, 4 (1983), pp. 312–316.
12. D. GHOSH, F. BRGLEZ, AND M. STALLMANN, *First steps towards experimental design in evaluating layout algorithms: Wire length versus wire crossing in linear placement optimization*, Tech. Report 1998-TR@CBL-11-Ghosh, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, October 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-TR@CBL-11-Ghosh>.

13. ———, *Hypercrossing number: A new and effective cost function for cell placement optimization*, Tech. 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>.
14. D. GHOSH, N. KAPUR, J. E. HARLOW, AND F. BRGLEZ, *Synthesis of Wiring Signature-Invariant Equivalence Class Circuit Mutants and Applications to Benchmarking*, in Proceedings, Design Automation and Test in Europe, Feb 1998, pp. 656–663. Also available at <http://www.cbl.ncsu.edu/publications/#1998-DATE-Ghosh>.
15. F. HARARY AND A. J. SCHWENK, *Trees with Hamiltonian Square*, *Mathematika*, 18 (1971), pp. 138–140.
16. ———, *A New Crossing Number for Bipartite Graphs*, *Utilitas Mathematica*, 1 (1972), pp. 203–209.
17. J. E. HARLOW AND F. BRGLEZ, *Design of Experiments in BDD Variable Ordering: Lessons Learned*, in Proceedings of the International Conference on Computer Aided Design, ACM, Nov. 1998. Also available from <http://www.cbl.ncsu.edu/publications/#1998-ICCAD-Harlow>.
18. M. JÜNGER AND P. MUTZEL, *2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms*, *Journal of Graph Algorithms and Applications (JGAA)*, 1 (1997), pp. 1–25.
19. F. T. LEIGHTON, *New Lower Bound Techniques for VLSI*, *Math. Systems Theory*, 17 (1984), pp. 47–70.
20. E. MÄKINEN, *A Note on the Median Heuristic for Drawing Bipartite Graphs*, *Fundamenta Informaticae*, 12 (1989), pp. 563–570.
21. ———, *Experiments on drawing 2-level hierarchical graphs*, *International Journal of Computer Mathematics*, 37 (1990), pp. 129–135.
22. M. MAREK-SADOWSKA AND M. SARRAFZADEH, *The Crossing Distribution Problem*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14 (1995), pp. 423–433.
23. P. MUTZEL, *The AGD-Library: Algorithms for Graph Drawing*, 1998. Available from <http://www.mpi-sb.mpg.de/mutzel/dfgdraw/agdlib.html>.
24. S. YANG, *Logic Synthesis and Optimization Benchmarks User Guide*, Tech. Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991. Now available from <http://www.cbl.ncsu.edu/publications/#1991-IWLS-UG-Saeyang> and benchmarks from <http://www.cbl.ncsu.edu/benchmarks/Benchmarks-upto-1996.html>.
25. F. SHAHROKHI, O. SÝKORA, L. A. SZÉKELY, AND I. VRT'Ů, *On bipartite drawings and the linear arrangement problem*, in *Lecture Notes in Computer Science*, no. 1467, Springer Verlag, 1997, pp. 55–68.
26. C. D. THOMPSON, *Area-Time complexity for VLSI*, in Proceedings, 11th Annual ACM Symposium on Theory of Computing, May 1979, pp. 81–88.
27. W. T. TUTTE, *Convex Representations of Graphs*, *Proc. London Math. Soc.*, 10 (1960), pp. 304–320.
28. ———, *How to Draw a Graph*, *Proc. London Math. Soc.*, 13 (1963), pp. 743–768.
29. J. N. WARFIELD, *Crossing Theory and Hierarchy Mapping*, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-7* (1977), pp. 505–523.

Appendix

Data set and programs used in our experiments are available on the Web at http://www.cbl.ncsu.edu/experiments/DoE_Archives/DoE_0003. The organization of this directory is outlined in Figure 9. Upon accessing the directory, researchers will find links and documentation to (1) reference circuits used to generate equivalence classes, (2) subdirectories of equivalence classes for each reference circuit (each class has 128 graphs), and (3) complete postings of results for treatments 0 to 19 as described in this paper.

For example, the `treatment_0019` archives results of experiments under Treatment 19 applied to each circuit class. The results of treatments for each class include (1) tables of crossing numbers for each graph instance in a *given class*; (2) statistics summary for each table such as mean, standard deviation, confidence interval for the mean, minimum and maximum; (3) tables of crossing numbers for each graph instance across *all classes*; (4) summary of crossing number statistics across all classes; (5) treatment-specific 2-layer graph orderings, archived for each equivalence class. Currently, orderings for Treatment 19 only are posted, since this treatment has generated node orderings with a crossing number mean that is consistently best across all equivalence classes.

The graph orderings, posted for Treatment 19, have been evaluated for a crossing number independently of the treatment that generated this order. The program we use is `cn_eval`, also available from the DoE-0003 archive. The program can be invoked with a simple command line

```
cn_eval graph.dot graph_trxxxx.ord
```

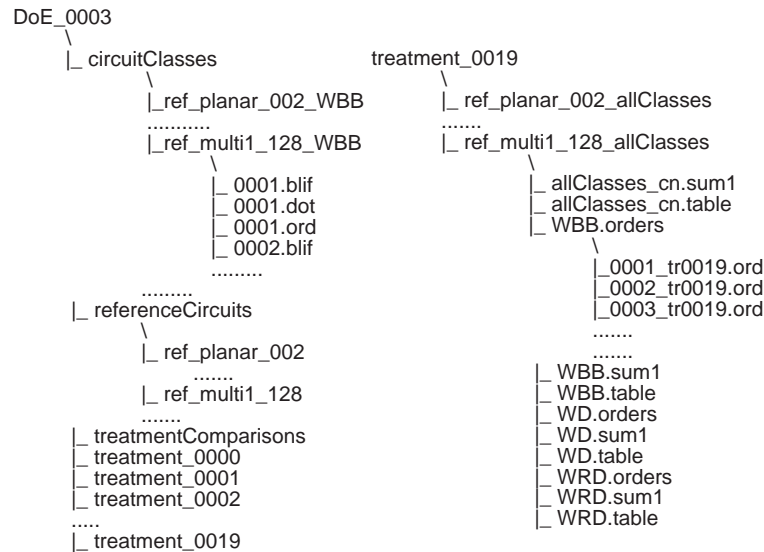


Fig. 9. DoE_0003 under http://www.cbl.ncsu.edu/experiments/DoE_Archives/

where `graph.dot` is an instance of the graph description in `dot` format, and `graph_trxxxx.ord` is the ordering associated with treatment `xxxx` (both formats are described briefly in this Appendix; `dot` format is also described on the Web [10]). The *initial orderings* for all instances of equivalence classes posted on the Web are associated with a random order, i.e. Treatment 0. The output of `cn_eval` is a single line of the form

```
graph_trxxxx crossings
```

where *crossings* is the number of crossings obtained when the graph is embedded using the given order.

The unique feature of our experimental design is thus the separation of the experiment into three parts: (1) the generation of experimental data (in this case programs that generate equivalence classes based on actual circuits), (2) the execution of heuristics, and (3) the evaluation of the cost function for the results produced by a heuristic. This separation of execution and evaluation makes it possible for other researchers to run their own heuristics on our data, submit new data to our heuristics, and evaluate any ordering independently with `cn_eval`.

Heuristics reported in this paper are implemented in the program `unfold2`, also available from the Web under the DoE_0003 archive. It is invoked with the command line

```
unfold2 -tr=N graph.dot order.ord
```

where `N` is the treatment number (as described in the paper). The extensions `.dot` and `.ord` are the expected extensions for graph description and layer ordering files, respectively. The program creates an output file called `graph_trxxxx.ord`, where `xxxx` is a 4-digit version of the treatment number (padded with 0's). This file specifies the ordering of the nodes on each layer as determined by the specified treatment.

Dot format. The expected input-file format for the `graph.dot` file is

```
digraph graph_name { statement; ... statement; }
```

where each statement defines an edge `layer0_node -> layer1_node`. Arbitrary white space is allowed between and within statements.

Ord format. An `.ord` file contains a sequence of layer descriptions separated by white space. Each layer description is of the form

```
layer_number { node_1 node_2 ... node_k }
```

where the layer number is an integer (0 or 1 in the case of bipartite graphs, but the notation can be generalized to more than 2 layers) and the `node_i`'s are names of nodes on the given layer. The current implementation requires that all nodes of the layer be present in the list and occur exactly once. The list specifies the left-to-right ordering of nodes on the given layer.

Anything between a `#` and the end of the line is interpreted as a comment. Arbitrary white space and/or comments can occur before the `{`, between nodes, or on either side of a layer description.