Crossing Reduction by Windows Optimization

Thomas Eschbach[1], Wolfgang Günther[2], Rolf Drechsler[3], and Bernd Becker[1]

[1] Institute for Computer Science
Albert-Ludwigs-University, Freiburg, Germany
{eschbach,becker}@informatik.uni-freiburg.de
[2] Infineon AG
CL DAT DF LD V, Munich, Germany
wolfgang.guenther@infineon.com
[3] Institute for Computer Science
University of Bremen, Bremen ,Germany
drechsle@informatik.uni-bremen.de

Abstract. The number of edge crossings is a commonly accepted measure to judge the "readability" of graph drawings. In this paper we present a new algorithm for high quality multi-layer straight-line crossing minimization. The proposed method uses a local optimization technique where subsets of nodes and edges are processed exactly. The algorithm uses optimization on a window applied in a manner, similar to those used in the area of formal verification of logic circuits. In contrast to most existing heuristics, more than two layers are considered simultaneously. The algorithm tries to reduce the total number of crossings based on an initial placement of the nodes and can thus also be used in a post-processing step. Experiments are given to demonstrate the efficacy of the proposed technique on benchmarks from the area of circuit design.

1 Introduction

In the last few years the area of automated graph drawing has received a lot of attention from academic and industrial researchers as well. Many relations arising in computer science and in other areas such as chemistry can be modeled by graphs. Concerning computer science visualization of graphs has a growing number of applications e.g. in the areas of data bases, software engineering, graphical interfaces, production scheduling and VLSI CAD (Very Large Scale Integration Computer Aided Design). To allow humans to quickly and efficiently interpret information given as graphs, a good drawing is very useful. In the following, we are especially interested in the drawing of digital circuits occurring in VLSI CAD applications, which can be modeled by graphs. Circuit designers need a fast tool to visualize the given circuit to get a good understanding of the design or diagnose bugs. Furthermore, highest drawing quality is needed for public presentation or documentation in this field. Here, the drawing quality is much more important than the required computation time.

Graph drawing - from the initial graph to the final layout - is usually split into several sub-tasks (for an overview of graph drawing see [1]). One very important step in the overall flow of graph drawing is *crossing minimization*, (see e.g. [5, 20, 21]). Unfortunately, even minimizing edge crossings in graphs with only two layers is *NP-hard* [7] and remains *NP-hard* even if the positions of the nodes in one layer are fixed. Several heuristic methods have been developed [4, 8, 12, 15, 18, 20], but exact solutions can be found only for small graphs [10, 11]. Almost all heuristic methods are based on the so-called *layer-by-layer sweep*:

- First, an initial order is chosen for all nodes. Then the positions of all nodes in the first layer are fixed and the positions of all nodes in the second layer are computed with respect to all nodes in the first layer.
- In the next step the algorithm fixes the positions of the nodes in the second layer and computes the positions of the nodes located in the third layer with respect to the second layer.
- The algorithm continues until the positions of the nodes in the last layer are computed followed by a sweep in the reverse order of layers from the last layer to the first layer.

Typically the procedure given above is iterated until no reduction in the number of crossings can be achieved in an iteration. By this method the multi-layer crossing minimization problem is reduced to the minimization of the edge crossings in the corresponding two-layer graphs by reordering only the nodes of one layer. Due to the significance of the problem, several heuristics have been proposed, e.g. [4, 15, 20, 21, 18]. In particular the *barycenter* method [20] is known for computing good solutions in a short time. Somewhat surprising is that using it in the same layer-by-layer sweep framework yields slightly better results than using a one-sided exact algorithm [11].

In [12] the crossing reduction is solved with tabu search, which usually computes good solutions at the expense of run time. In [15, 8] *sifting* has been proposed, a heuristic that was originally used for minimizing the number of nodes in *Binary Decision Diagrams (BDDs)* used frequently in logic synthesis applications and formal verification of logic circuits. In the BDD context, a technique called *window optimization* [6, 9] has been proposed that facilitates smooth trade-off of run time for quality.

In this paper we propose the use of windows optimization for multi-layer crossing minimization. It is designed for post-processing a given solution in the area of circuit visualization, if the corresponding graph is too large for an exact approach. An initial order, e.g. computed by one of the approaches described above is improved in the following way: A series of subsets of nodes with constant size typically spreading over several layers are extracted and optimized exactly with respect to their adjacent nodes. The exact method makes use of lower bound computations to prune the search space. Only if the local solution induces a crossing reduction of the whole graph the order of all nodes is re-adjusted. The designer can smoothly influence the trade-off between run time and solution quality by choosing the size of the window.

The implementation of the algorithms is done in *AGD* [19], a state-of-the-art library of algorithms for graph drawing. Since most of the algorithms discussed above are implemented in AGD, a fair comparison is possible. The *AGD* uses the efficient *LEDA* graph package for combinatorial and geometric computing [17]. Our experiments show that further reductions of up to 10% compared to the *AGD* results can be obtained by using the new technique.

The paper is organized in the following way. First we introduce the basic definitions in the field of the multi-layer straight-line crossing minimization problem. Then a brief outline of the exact algorithm used is given. In the following section the new approach is presented. Next we give the experimental results showing the crossing reduction obtained by applying the windows optimization procedure on the graphs already optimized with the *AGD* package.

2 Preliminaries

A directed graph $G = (V, E)$ is a *multi-layered* graph with d layers if the node set V is partitioned into d subsets $V_1, V_2, \ldots, V_d$, i.e. $V_1 \cup V_2 \cup \ldots \cup V_d = V$ and $(\forall m \neq m')\ V_m \cap V_{m'} = \emptyset$. V_m is called the *m-th layer* of the graph. All edges in E connect nodes in different layers. A layering of a graph is called *proper* if the edges only connect nodes of adjacent layers V_m and V_{m+1}. If a layering of a graph is not proper one can introduce *dummy nodes* along edges (u, v) for which $l = \text{layer}(v) - \text{layer}(u) > 1$. We replace (u, v) by a path $(u = v_1, v_2, \ldots, v_l = v)$ of length l. In each layer between u and v, one dummy node has to be placed. Algorithms for layering graphs which represent circuits are discussed e.g. in [3]. It is important to note that only the order of the nodes in a layer V_m, $m \in \{1, \ldots, d\}$, affects the number of crossings with adjacent layers. To solve the exact multi-layer straight-line crossing minimization problem we have to determine for all layers m an order ord_m containing all nodes in layer V_m so that the number of crossings is minimized. In the following a set of orders ord_m, $m \in \{1, \ldots, d\}$, is called an order for the graph.

We define $x_{ij}^m = 1$ if $ord_m(i) < ord_m(j)$, 0 otherwise. It is easily seen, that for a given order of a graph with d layers the number of crossings can be expressed by:

$$C(order) = \sum_{m=1}^{d-1} \sum_{i=1}^{|V_m|-1} \sum_{k=i+1}^{|V_m|} \sum_{j \in N(i)} \sum_{l \in N(k)} (x_{ik}^m \cdot x_{lj}^{m+1} \;+\; x_{ki}^m \cdot x_{jl}^{m+1}) \qquad (1)$$

where $N(u) = \{v \in V \mid e = (u, v) \in E\}$ denotes the set of neighbors of $u \in V$.

2.1 Exact Algorithm

We briefly sketch the algorithm for the multi-layer straight-line crossing minimization problem. The algorithm computes the exact optimum and makes use of a lower bound technique to reduce the search space. The main idea goes back to dynamic programming.

- The algorithm computes the minimum number of edge crossings for all nodes by considering the solutions for all subsets of nodes. In the first step we consider all subsets S with one node. For each subset the algorithm creates an entry I and stores it in a table. Each entry I contains the number of edge crossings caused by the edges with both ends in S. Additionally the algorithm stores the permutation of the nodes within S in every entry I. In the first step the first node is also the first node in the permutation of the nodes in S

- An initial upper bound of edge crossings is obtained by using the barycenter heuristic method to compute an order for all nodes [20]. Let $S(I)$ denote the set of nodes stored in the entry I. In the second step we process every entry I of the table in the following way: For each node n_i, with $n_i \notin S(I)$, we create a new entry I_{new} for $S(I) \cup n_i$ and store it in the new table if the upper bound is larger than the number of crossings of the new subset $S(I_{new})$. Then the algorithm enlarges the permutation of I_{new} with n_i and updates the number of crossings.
- If there are two table entries with the same subsets of nodes, the one with the larger number of crossings can be deleted, if the following holds: there exists no more than one edge per layer, that is connected to a node u outside $S(I)$, except that they point to different layers. Otherwise, we cannot compute the number of crossings induced by the permutation of the nodes in the subset $S(I)$.
- To prune the search space the following lower bound L is used: We count the number $|u|$ of edges which leave I from layer m for nodes in layer $m+1$ (not in I) and the number $|v|$ of edges which leave I in layer $m+1$ for nodes in layer m not in I. Then L can be computed as the number of crossings in I plus the product of $|u|$ and $|v|$. If L is larger than the upper bound we can also delete the table entry.
- To speed up the algorithm a dynamic hash table is used to store the current permutation together with the current lower bound. For each entry I the corresponding set of nodes is used as a hash key.

In the following we embed this exact algorithm in our windows optimization scenario. However it should be noted that, in principle, the optimization technique described below also works in combination with other exact approaches, like e.g. [11].

3 Windows Optimization

The basic idea is to extract subsets S of nodes from the graph with constant size and the corresponding subset of edges with at least one end in S. Then the optimal order for S and a somewhat simplified representation of neighborhood N of S is computed. (The notion simplified neighborhood is explained below.) If the exactly optimized order of the nodes in S with regard to N reduces the number of crossings of the graph we adopt the locally optimized solution, otherwise we proceed with a new subset S.

More precisely, the algorithm extracts windows of D layers with a maximum of W nodes in each layer[1]. First, we extract windows with nodes located in the upper D layers. Starting at the leftmost nodes occurring in the current global permutation the algorithm slides the window one position to the right after each attempt to reduce the number of crossings. We compute a sequence of windows whose horizontal size is adapted layer by layer based on the layers with the largest number of nodes. Then the window processes the following D layers, i.e. layers two to layer $D+1$, in the same manner until we have traversed the whole graph. We repeat the procedure described above until no further reduction of the number of crossings can be achieved. Then we may increase the number of layers which are considered by incrementing D and repeat the windows optimization procedure to reduce the number of crossings. This allows to achieve a smooth trade-off between quality and run time. A sketch of the algorithm is given below:

```
WindowsOptimization(Graph G, Order Ord, int D, int W) {
    C_old = #crossings(G, Ord)
    d = number of layers in G
    repeat {
        C_new = C_old
        for(i = 1; i ≤ d − D + 1) {
            k = max_{i≤m<i+D} (nodes in layer m)
            for(j = 1; j ≤ k − W) {
                I = all nodes n located in layer i,..., layer i + D − 1 with
                            j ≤ Ord(n) ∗ k/(#nodes in n's layer) < j + W
                N = compute neighborhood of I
                set G to order Ord
                minimize exactly(I, N)
                Ord ' = current Order
                if(#crossings(G, Ord ') <  C_new) {
                    C_new = #crossings(G, Ord ' )
                    Ord = Ord '
                }
            } // end for
        } // end for
    } until (C_new = C_old)
    return Ord
}
```

All nodes $n \notin S$, where n is connected to S via one or more edges, are called the neighborhood N of S. We simplify the representation of the neighborhood to avoid long run times of the exact algorithm as follows. First we ignore all

[1] In the experiments we initialize the values D with two and W to four.

crossings produced by edges which are connected to nodes located in the layer below all nodes of S. They will be considered when processing the next level. All edges which are connected to nodes located in the layer above all nodes of S and which are not connected to a node $n \in S$ are also ignored. A boundary node is created for each node in the layer that is above all nodes in S if at least one of its edges is connected to a node in S. It is important to notice that the given graph is converted to a proper graph by adding dummy nodes and hence the layer above all nodes in the subset S is uniquely defined. Edges which are connected to nodes in a layer which all point in the same direction with respect to the current order of the graph are redirected to a single boundary node located in the same layer and in the same direction. In other words, the problem is solved using the exact algorithm to compute the optimal order of S with respect to a simplified representation of the neighborhood. An example given below illustrates an instance of the simplified representation of the neighborhood.

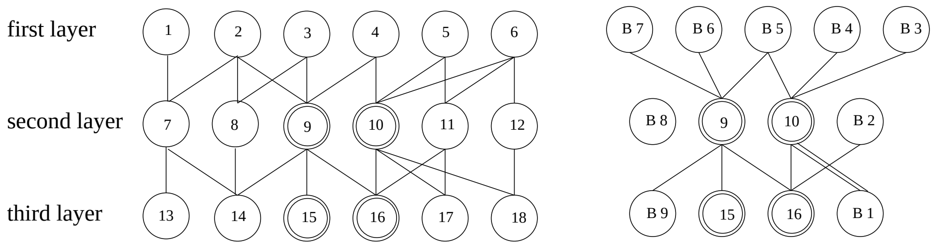

Fig. 1. Subset {9,10,15,16} and the corresponding boundary nodes

Example 1. The left side of Figure 1 shows an example graph. The algorithm tries to reduce the edge crossings by optimizing the order of the nodes in the subset S *({9,10,15,16}). First the algorithm computes a simplified representation of the neighborhood (shown in the right side of Figure 1). Node* 1 *can be ignored, because it is not connected to a node in* S*. The node sets {7,8}, {11,12}, {13,14}, and {17,18} are each collapsed to single boundary nodes B8,B2,B9 and B1 respectively. The procedure computes the optimal order of* S *with respect to this neighborhood.*

4 Experimental Results

The algorithm has been implemented in C and integrated in AGD. All experimental results are based on graphs which are extracted from the sequential benchmark circuits in [2, 16]. The circuits are modeled as proposed in [14] by creating a node for every input, every output and every gate of the circuit. Also one has to create a fanout node for every gate n_i which has more than one fanout branch. The fanout node is conected with the output of n_i and the inputs of the corresponding gates. A depth first search assigned each node to one level, starting at the input nodes. Also dummy nodes were created to compute a proper layering of the graph. All experiments were carried out by running the implemented procedures on a 900 MHz personal computer with 500 MB main memory with linux OS. All run times are given in CPU seconds.

For comparison we utilized the AGD [19] barycenter, median, weighted-median, split and sifting implementation all combined with the *greedy switch* method after each traversal of the graph. It turned out, that the barycenter method combined with the *greedy switch* method clearly dominates the others (see also [11]). In Figure 2 we illustrate the crossing reduction in percent referenced to the barycenter method. The x-axis corresponds to different benchmarks and on the y-axis we give the number of crossings obtained using different procedures relative to the number of crossings obtained using barycenter procedure. Due to these results, we restrict ourselves to a comparison of the proposed procedure with barycenter *(bc)* in combination with greedy switch *(bc+gs)* in the following.

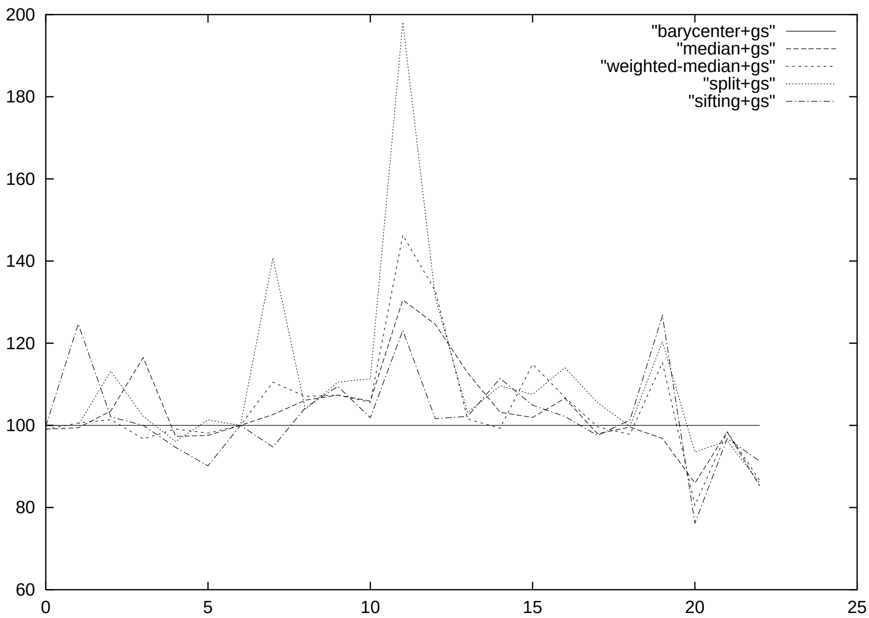

Fig. 2. Crossing reduction compared to barycenter+greedyswitch.

In Table 1 the results are given. In the second column the number of nodes of each graph is shown. In the third and the fourth column the number of edge crossings using AGD barycenter implementation and the barycenter implementation combined with the greedy switch method are given, respectively. On average 13% less crossings are obtained with *bc+gs*. This demonstrates that in the VLSI CAD scenario it is very effective to use the *greedy switch* method. We further post-processed the order obtained by *bc+gs* method with our windows optimization algorithm shown in the column *wo+bc+gs*. In almost all cases the number of crossings could be further reduced. On average the number of crossings went down by 4% and in some cases we even obtained a reduction of more than 10%. We also made experiments running more iterations of the layer-by-layer based methods. Spending the same amount of time as for window optimization usually dose not lead to better results.

Table 1. Benchmark results

circuit	nodes	bc	$bc + gs$	$wo + bc + gs$	time
add6	504	230	170	159	30
alu1	125	101	91	86	5
alu2	391	602	558	536	52
alu3	452	791	742	721	43
adr4	210	186	144	137	10
co14	261	97	76	69	13
dk17	437	444	440	435	37
dk27	186	111	95	95	4
dk48	641	619	591	582	67
mish	571	127	108	107	11
radd	215	68	61	54	5
rd53	193	371	320	290	27
s208	665	385	280	269	65
s298	638	1312	1142	1119	42
s382	903	1288	1199	1143	204
s386	763	4243	3868	3723	254
s400	941	1380	1366	1342	77
vg2	456	241	157	149	27
x1dn	470	218	185	179	43
x9dn	419	209	188	183	24
z4	238	184	149	140	17
Z9sym	767	5300	5189	4917	414

Another advantage of the proposed method is that it allows to smoothly trade-off run time versus quality. A trade-off between crossing reduction and run time consumption can be achieved if we start with a small window and successively increment the number of affected layers. More precisely, at the beginning we adjust the window to observe two layers with four nodes in each layer. If no crossing reduction could be achieved after a traversal of the graph we enlarge the window adding one layer until we reach a depth of four layers. In Figure 3 the effect of the process is demonstrated by an experiment. In Figure

3 also the *bc+gs* procedure gives the 100%-line. It can be easily seen that with increasing window size the quality is significantly improved.

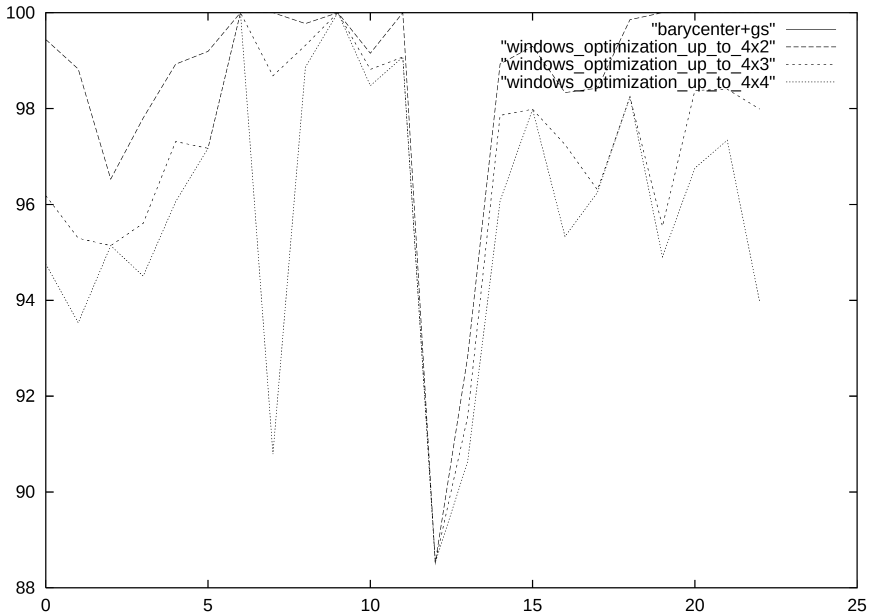

Fig. 3. Trade-off between quality and run time

5 Conclusions

We presented a new method to improve the results of layer by layer sweep crossing minimization algorithms. The algorithm changes the permutation of the nodes if the local optimization leads to a reduction of the number of crossings. The experiments have shown that this method can enhance the quality of already locally optimized solutions produced by AGD by more than 10%. In the average case a crossing reduction of four percent could be observed. If graph drawings are used for documentation in VLSI CAD a small improvement is of significant value.

Future work is directed towards improving the run time of the algorithm. For this, alternative implementations of the exact method [10, 11] will be studied instead of the one used here.

6 Acknowledgment

We are grateful to professor Sudhakar Reddy, University of Iowa, for fruitful discussions.

References

1. G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for Drawing Graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 4:235-282,1994.
2. F. Brglez, D. Bryan, and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits. *Int'l Symp. Circ. and Systems,* 1924-1934,1989.
3. R. Drechsler, W. Günther, L. Linhard, and G. Angst. Level Assignment for Displaying Combinational Logic. In *EUROMICRO*, 148-151,2001.
4. P. Eades, and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combin.,* 21.A:89-98,1986.
5. P. Eades, and K. Sugiyama. How to draw a Directed Graph. *Journal of Information Processing*, Vol.13,4,424-437,1991.
6. M. Fujita, Y. Matsunaga, and T. Kakuda. On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Level Synthesis. *European Conf. on Design Automation*, 50-54,1991.
7. M. R. Garey, and D.S. Johnson. Crossing number is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods,* 4:312-316,1983.
8. W. Günther, R. Schönfeld, B. Becker, and P. Molitor. K-Layer Straightline Crossing Minimization by Speeding up Sifting. *Proceedings of the 8th International Symposium on Graph Drawing*, 253-258,2000.
9. N. Ishiura, H. Sawada, and S. Yajima. Minimization of Binary Decision Diagrams Based on Exchange of Variables. *Int'l Conf. on CAD*, 472-475,1991.
10. M. Jünger, E.K. Lee, P. Mutzel, and T. Odenthal. A Polyhedral Approach to the Multi-Layer Crossing Number Problem. *Proceedings of the 5th International Symposium on Graph Drawing*, LNCS Vol. 1353,13-24,1997.
11. M. Jünger and P. Mutzel. 2-Layer Straightline Crosing Minimization: Performance of Exact and Heuristic Algorithms. *J. Graph Agorithms Appl.,* 1(1):1-25,1997.
12. M. Laguna, R. Martí, and V. Valls. Arc Crossing Minimization in Hierarchical Digraphs with Tabu Search. *Computers and Operations Research*, Vol. 24,2:1175–1186,1997.
13. E. Mäkinen. Experiments on drawing 2-level hierarchical graphs. *International Journal of Computer and Mathematics*, 36:175-181,1990.
14. E. Mäkinen. How to draw a hypergraph. *International Journal of Computer and Mathematics*, 34:177-185,1990.
15. C. Matuszewski, R. Schönfeld, and P. Molitor. Using Sifting for k-Layer straightline crossing minimization. *Proceedings of the 7th International Symposium on Graph Drawing*, LNCS 1731,217-224,1999.
16. K. McElvain Benchmark Set: Version 4.0. *International Workshop on Logic Synthesis,* 1993.
17. K. Mehlhorn, and S. Näher. LEAD: A Platform for Combinatorial and Geometric Computing. *Communications of the ACM*, 38,1:96-102,1995.
18. P. Mutzel. An Alternative Method for Crossing Minimization on Hierarchical Graphs. *SIAM Journal on Optimization*,11,4, 1065-1080,2001.
19. P. Mutzel, C. Gutwenger, R. Brockenauer, S. Fialko, G. W. Klau, M. Krüger, T. Ziegler, S. Näher, D. Alberts, D. Ambras, G. Koch, M. Jünger, C. Buchheim, and S. Leipert. AGD: A Library of Algorithms for Graph Drawing. *Proceedings of the 6th International Symposium on Graph Drawing*, LNCS 1547,456-457,1998.
20. K. Sugiyama, S.Tagawa, and M.Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transaction on Systems, Man and Cybernetics,* 11(2):109-125,1981.
21. J. Warfield. Crossing Theory and Hierarchy Mapping. *IEEE Transactions on System, Man, and Cybernetics,* SMC-7,7,502-523,1977.