Fixed Linear Crossing Minimization by Reduction to the Maximum Cut Problem*

Christoph Buchheim[1] and Lanbo Zheng[2,3]

[1] Computer Science Department, University of Cologne, Germany
buchheim@informatik.uni-koeln.de
[2] School of Information Technologies, University of Sydney, Australia
[3] IMAGEN program, National ICT Australia
lzheng@it.usyd.edu.au

Abstract. Many real-life scheduling, routing and location problems can be formulated as combinatorial optimization problems whose goal is to find a linear layout of an input graph in such a way that the number of edge crossings is minimized. In this paper, we study a restricted version of the linear layout problem where the order of vertices on the line is fixed, the so-called fixed linear crossing number problem (FLCNP). We show that this $\mathcal{NP}$-hard problem can be reduced to the well-known maximum cut problem. The latter problem was intensively studied in the literature; efficient exact algorithms based on the branch-and-cut technique have been developed. By an experimental evaluation on a variety of graphs, we show that using this reduction for solving FLCNP compares favorably to earlier branch-and-bound algorithms.

1 Introduction

For a given simple graph $G = (V, E)$ with vertex set V and edge set E, a linear embedding of G is a special type of embedding in which vertices of V are placed on a horizontal line L and edges are drawn as semicircles above or below L; see Fig. 1. This type of drawing was first introduced by Nicholson [12] in order to develop a heuristic algorithm for the general $\mathcal{NP}$-complete crossing minimization problem [4]. However, Masuda et al. proved that it is still $\mathcal{NP}$-hard to find a linear embedding of a given graph with a minimum number of crossings, even if the ordering of vertices on L is predetermined [10]. The latter problem is called the *fixed linear crossing number problem* (FLCNP).

Crossing minimization for linear embeddings has important applications in different areas such as sorting permutations [6], fault tolerant VLSI design [13], complexity theory [3], and compact graph encodings [11]. Moreover, the problem FLCNP is of general interest in graph drawing and information visualization, where the number of edge crossings has a big effect on the readability of graph layout [2]. It was also shown to be a subproblem in communications network management graphics facilities such as CNMgraf [5]. Sorting with parallel stacks

* Partially supported by the Marie Curie RTN ADONET 504438 funded by the EU.

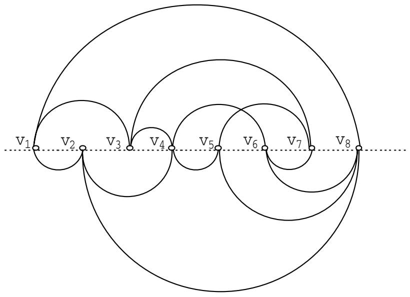

Fig. 1. A linear embedding

is similar to FLCNP where the layout of vertices is fixed, although the objective is to find a layout with no crossings at all.

Recently, heuristic methods, as well as exact algorithms, have been proposed to find optimal or near-optimal solutions of linear layout problems. Cimikowski [1] presented different powerful heuristics as well as an exact branch-and-bound algorithm for FLCNP. In the worst case, the latter enumerates all possible assignments of edges to the two sides of L (up to symmetry). However, the idea of branch-and-bound is to use known bounds on the objective function in order to skip most feasible solutions during the enumeration. Cimikowski's algorithm is able to find exact solutions for graphs with up to 50 edges.

In this paper, we introduce a new exact algorithm for the problem FLCNP that is based on a reduction to the *maximum cut problem* (MAXCUT). The same reduction yields a simple test for fixed linear planarity. Computational results for our approach are compared to those obtained with the exact algorithm of [1], on exactly the same test data and equipment. Our approach yields a remarkable improvement in terms of computational efficiency.

This paper is organized as follows. In Sect. 2, the problem under consideration is formalized and necessary notation is introduced. In Sect. 3, we describe the reduction from FLCNP to MAXCUT and present a corresponding optimization algorithm. Experimental results are analyzed in Sect. 4, and Sect. 5 concludes.

2 Preliminaries

Throughout this paper, we consider an undirected, simple graph $G = (V, E)$ with vertex set V and edge set E. A *vertex ordering* (or *vertex permutation*) of G is a bijection $\delta \colon V \to \{1, 2, \ldots, |V|\}$. For a pair of vertices (v, w), we will shortly write $v < w$ instead of $\delta(v) < \delta(w)$.

In a *fixed linear embedding* of G, we assume that the vertices of G are placed on a straight horizontal line L according to a fixed vertex ordering. Moreover, each edge is drawn as a semicircle; see Fig. 1. Consequently, edges may be routed above or below L but never cross L. Notice that three edges cannot intersect in one point unless it is a common endpoint.

For a given graph G and vertex ordering δ, a pair of edges $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$ is *potentially crossing* if e_1 and e_2 cross each other when routed on the same side of L. Clearly, e_1 and e_2 are potentially crossing if and only if $v_1 < v_2 < w_1 < w_2$ or $v_2 < v_1 < w_2 < w_1$.

In this paper, we are interested in the number of crossings in fixed linear embeddings of G. There is a crossing between e_1 and e_2 if, and only if:

– e_1 and e_2 are potentially crossing, and
– e_1 and e_2 are embedded on the same side of L.

We are going to address the following optimization problem:

Fixed linear crossing number problem (FLCNP): Given a graph $G = (V, E)$ with a fixed vertex ordering, find a corresponding linear embedding of G with a minimum number of edge crossings.

It is easy to see that the number of edge crossings in a linear embedding only depends on the order of vertices and the sides to which the edges are assigned, but not on the exact positions of the vertices. In particular, as the vertex ordering is fixed as part of the input of FLCNP, the only remaining choice is whether edges are drawn above or below the line L. Thus, with respect to the crossing number, there are essentially $2^{|E|}$ different fixed linear embeddings of G. Nevertheless, the problem FLCNP was shown to be $\mathcal{NP}$-hard by Masuda et al. [10].

3 A New Algorithm

The exact algorithm used by Cimikowski [1] to solve the problem FLCNP is based on the branch-and-bound technique: basically, all possible solutions of the problem are enumerated. The set of solutions is given by a binary enumeration tree, where each inner node corresponds to a decision whether a chosen edge is drawn above or below the horizontal line. In the worst case, an exponential number of solutions has to be enumerated. However, the basic idea of branch-and-bound is the pruning of branches in this tree: at some node of the tree, a certain set of edges is already fixed. According to this information, one can derive a lower bound on the number of crossings subject to these fixed edges. If this lower bound is at most as good as a feasible solution that has already been found, e.g., by some heuristics, it is clear that the considered subtree cannot contain a better solution, so it does not have to be explored.

In the following, we describe a different approach for solving FLCNP exactly. We show that the problem is, in fact, a special case of the well-known MAXCUT problem; see Sect. 3.1. The latter has been studied intensively in the literature. In particular, branch-and-cut algorithms have been developed; see Sect. 3.2, which we use for our experimental evaluation presented in Sect. 4.

3.1 Reduction to MAXCUT

In this section, we show that the problem FLCNP can easily be reduced to the maximum cut problem given as follows:

Maximum Cut Problem (MAXCUT): Given an undirected graph $G' = (V', E')$, find a partition of V' into disjoint sets V_1 and V_2 such that the number of edges from E' that have one endpoint in V_1 and one endpoint in V_2 is maximal.

For an instance of FLCNP, i.e., a given graph $G = (V, E)$ with a fixed vertex permutation, we construct the associated *conflict graph* $G' = (V', E')$ as follows: the vertices of G' are in one-to-one correspondence to the edges of G, i.e., $V' = E$. Two vertices of G' corresponding to edges $e_1, e_2 \in E$ are adjacent if, and only if, e_1 and e_2 are potentially crossing. See Fig. 2 for an illustration.

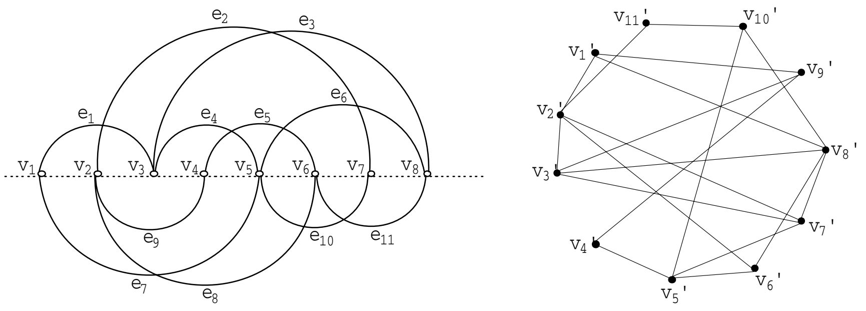

Fig. 2. The graph G and its associated conflict graph G'

Definition 1. *Let G be a graph with a fixed vertex permutation. Given a vertex partition (V_1, V_2) of its conflict graph G', the associated* cut embedding *is the fixed linear embedding of G where edges corresponding to V_1 and V_2 are embedded to the half spaces above and below the vertex line, respectively.*

Theorem 1. *Consider a partition (V_1, V_2) of V'. Then the corresponding cut embedding is a fixed linear embedding of G with a minimum number of crossings if, and only if, (V_1, V_2) is a maximum cut in G'.*

Proof. Let F' be the set of edges in G' with one endpoint in V_1 and one endpoint in V_2, i.e., the cut given by (V_1, V_2). By definition of G', we know that every crossing in the cut embedding associated to (V_1, V_2) corresponds to an edge in G' such that either both its endpoints belong to V_1, or both belong to V_2, i.e., to an edge in $E' \setminus F'$. Thus, the number of crossings is $|E'| - |F'|$. As $|E'|$ is constant for a fixed vertex permutation, the result follows. $\square$

Theorem 2. *For a graph $G = (V, E)$ with a fixed vertex permutation, there is a planar fixed linear embedding of G if, and only if, the associated conflict graph G' of G is bipartite.*

Proof. Suppose H is a planar fixed linear embedding of G. Let E_1 and E_2 represent the two edge sets above and below the horizontal vertex line, respectively. Then the vertices of G' consist of two vertex sets V_1 corresponding to E_1 and V_2

corresponding to E_2. Since H is planar, there is no edge connecting vertices from the same set. So G' is bipartite. On the other hand, if G' is bipartite, the resulting cut embedding of G is obviously planar. □

Observe that testing whether the graph G' is bipartite can be done in linear time (with respect to G') by two-coloring a DFS-tree.

3.2 Solving MAXCUT by Branch-and-Cut

By Theorem 1, we can use any algorithm for MAXCUT in order to solve FLCNP. One of the most successful approaches for solving MAXCUT to optimality in practice is branch-and-cut.

It would go beyond the scope of this paper to explain this approach in detail. Roughly, the problem is modelled as an integer linear program (ILP). This ILP is first solved as a linear program (LP), i.e., the integrality constraints are relaxed. LPs are solved very quickly in practice. If the LP-solution is integer, we can stop. Otherwise, one tries to add cutting planes that are valid for all integer solutions of the ILP but not necessary for (fractional) solutions of the LP. If such cutting planes are found, they are added to the LP and the process is reiterated.

We have to resort to the branching part only if no more cutting planes are found. In general, only a small portion of the enumeration tree has to be explored, as many branches can be pruned. Compared to a pure branch-and-bound approach as presented in [1], the number of subproblems to be considered is very small in general. This, however, depends on the quality of the cutting planes being added. The latter in turn depend on the specific problem; finding good cutting planes is a sophisticated task. Fortunately, the MAXCUT problem has been investigated intensively, so that many classes of cutting planes are known.

More detailed information on algorithms for MAXCUT using cutting plane techniques can be found in [7,9]. Observe that MAXCUT can also be adressed by semidefinite programming methods; see e.g. [8]. These methods perform well on very dense instances, while being outperformed by ILP approaches on sparse or large graphs. For this reason, we chose the latter method for our experiments.

4 Experimental Results

In order to evaluate the practical performance of our new exact approach to FLCNP presented in the previous section, we performed extensive experiments. In this section, we report the results and compare them to the results obtained with the branch-and-bound algorithm proposed by Cimikowski [1]. The set of test instances is exactly the same as used in [1].

These instances mainly arise from network models of computer architectures; in general they are hamiltonian. The fixed order of nodes, as part of the input of FLCNP, is then determined by a hamiltonian cycle in the graph, as an ordering of the vertices along a hamiltonian cycle tends to yield a smaller number of crossings in general. In our experiments, we always used the same ordering as chosen in [1] for ensuring comparability.

More specifically, the networks considered are the following, see also [1]:

– *complete graphs* K_n for $n = 5, \ldots, 13$
– *hypercubic networks*: this class of graphs includes the *hypercubes* Q_d and several derivatives of hypercubes such as the *cube-connected-cycles* CCC_d, the *twisted cubes* TQ_d, the *crossed cubes* CQ_d, the *folded cubes* FLQ_d, the *hamming cubes* HQ_d, the *binary de Bruijn graphs* DB_d and the *undirected de Bruijn graphs* UDB_d, the *wrapped butterfly graphs* WBF_d and the *shuffle-exchange graphs* SX_d
– other interconnection networks, including the $d \times d$ *tori* $T_{d,d}$, the *star graphs* ST_d, the *pancake graphs* PK_d, and the *pyramid graphs* PM_d
– *circular graphs*: the circular graph $C_n(a_1, \ldots, a_k)$ is regular and hamiltonian.

In Table 1, we contrast our runtime results with those of the branch-and-bound algorithm presented in [1]; we list all instances for which runtimes are reported in [1]. For a better comparison, we ran both algorithms on the same machine, a Pentium 4 with 2.8 GHz. The running times for the branch-and-bound algorithm were obtained with the original implementation used in [1]. In the remainder of this section, all running times are given in CPU seconds.

Table 1. Running times for exact approaches

instance	B & B [1]	MAXCUT
Q_4	0.01	0.00
CCC_3	0.02	0.00
SX_4	0.01	0.00
FLQ_4	0.13	0.42
UDB_5	0.43	0.07
$C_{26}(1,3)$	0.46	0.00
$T_{6,6}$	1.27	0.04
CCC_4	2.59	0.01
K_{10}	2.27	3.21
SX_5	2.16	1.84
$C_{20}(1,2,3)$	16.69	0.39
$T_{7,7}$	64.89	0.15
$C_{22}(1,2,3)$	73.16	0.39
K_{11}	148.21	24.56
Q_5	612.35	1.67
K_{12}	1925.51	79.15
K_{13}	> 86400.00	2119.12

Notice that in our approach we did not use any initial heuristics, in order to give clearer and independent runtime figures. Nevertheless, as obvious from Table 1, our approach is much faster than the branch-and-bound algorithm. This is particularly true for sparse instances, e.g., Q_5. However, our approach outperforms [1] also on the larger complete graphs.

For all other instances, only heuristic results are given in [1]. Tables 2 and 3 state the results of our approach, sorted as in [1]. In all tables, the columns

Table 2. Experimental results, part I

instance	best heuristic [1]	worst heuristic [1]	exact solution	MAXCUT runtime
K_5	1	1	1	0.00
K_6	3	4	3	0.00
K_7	9	11	9	0.01
K_8	18	24	18	0.06
K_9	36	46	36	0.83
K_{10}	60	80	60	3.21
K_{11}	100	130	100	24.56
K_{12}	150	200	150	79.15
K_{13}	225	295	*225*	2119.12
Q_4	8	8	8	0.00
Q_5	62	80	**60**	1.67
Q_6	370	512	**368**	17924.24
Q_7	1874	2688	[1894]	> 86400.00
CCC_3	0	4	0	0.00
CCC_4	16	24	16	0.01
CCC_5	104	148	*104*	3.99
SX_4	7	8	7	0.00
SX_5	60	74	60	1.84
SX_6	281	333	[285]	> 86400.00
SX_7	1315	1554	[1319]	> 86400.00
UDB_4	5	5	5	0.00
UDB_5	28	34	28	0.07
UDB_6	149	183	**148**	19.88
UDB_7	629	815	[646]	> 86400.00
UDB_8	2384	3065	[2387]	> 86400.00
FLQ_3	4	6	4	0.00
FLQ_4	36	44	36	0.42
FLQ_5	208	256	*208*	5981.78
FLQ_6	1036	1320	[1061]	> 86400.00
FLQ_7	4712	6144	[4804]	> 86400.00
TQ_3	1	1	1	0.00
TQ_4	8	10	8	0.00
TQ_5	65	83	**63**	1.42
TQ_6	372	516	*372*	28694.06
TQ_7	1866	2693	[1916]	> 86400.00
CQ_3	1	1	1	0.00
CQ_4	12	12	12	0.01
CQ_5	88	106	88	6.47
CQ_6	494	588	[508]	> 86400.00
CQ_7	2475	3056	[2481]	> 86400.00
HQ_3	5	6	5	0.00
HQ_4	50	57	50	1.86
HQ_5	303	361	[303]	> 86400.00
HQ_6	1523	1885	[1531]	> 86400.00
HQ_7	6913	8734	[7057]	> 86400.00
WBF_3	22	30	22	0.02
WBF_4	164	205	**158**	22.54
WBF_5	904	1066	[948]	> 86400.00

Table 3. Experimental results, part I

instance	best heuristic [1]	worst heuristic [1]	exact solution	MAXCUT runtime
$T_{3,3}$	3	4	3	0.00
$T_{4,4}$	8	8	8	0.00
$T_{5,5}$	20	30	20	0.02
$T_{6,6}$	24	38	24	0.04
$T_{7,7}$	48	70	48	0.15
$T_{8,8}$	48	80	48	0.16
$T_{9,9}$	88	142	*88*	0.71
$T_{10,10}$	80	190	*80*	0.69
ST_4	11	13	11	0.01
ST_5	570	699	[572]	$>$ 86400.00
PK_4	10	11	10	0.01
PK_5	500	564	[514]	$>$ 86400.00
PM_3	4	26	4	0.00
PM_4	439	796	*439*	28964.87
$C_{20}(1,2)$	0	4	0	0.00
$C_{20}(1,2,3)$	22	28	22	0.39
$C_{20}(1,2,3,4)$	70	98	*70*	1.49
$C_{22}(1,2)$	0	2	0	0.00
$C_{22}(1,2,3)$	24	32	24	0.39
$C_{22}(1,3,5,7)$	200	254	*200*	191.70
$C_{24}(1,3)$	12	16	12	0.00
$C_{24}(1,3,5)$	72	92	72	1.68
$C_{24}(1,3,5,7)$	216	282	*216*	266.11
$C_{26}(1,3)$	14	18	14	0.00
$C_{26}(1,3,5)$	82	102	*82*	22.79
$C_{26}(1,4,7,9)$	364	446	*364*	19392.85
$C_{28}(1,3)$	16	20	**14**	0.00
$C_{28}(1,3,5)$	86	110	*86*	3.38
$C_{28}(1,2,3,4)$	98	138	*98*	3.90
$C_{28}(1,3,5,7,9)$	560	714	[560]	$>$ 86400.00
$C_{30}(1,3,5)$	96	120	**90**	2.77
$C_{30}(1,3,5,8)$	302	348	[**298**]	$>$ 86400.00
$C_{30}(1,2,4,5,7)$	392	470	[396]	$>$ 86400.00
$C_{32}(1,2,4,6)$	160	202	*160*	21.65
$C_{34}(1,3,5)$	110	132	**104**	5.83
$C_{34}(1,4,8,12)$	574	670	[**572**]	$>$ 86400.00
$C_{36}(1,2,4)$	36	60	*36*	0.03
$C_{36}(1,3,5,7)$	328	422	*328*	5624.53
$C_{38}(1,7)$	84	98	84	14.70
$C_{38}(1,4,7)$	190	236	*190*	149.04
$C_{40}(1,5)$	56	64	56	5.04
$C_{42}(1,4)$	42	46	42	0.07
$C_{42}(1,3,6)$	158	170	**150**	651.18
$C_{42}(1,2,4,6)$	210	284	*210*	115.16
$C_{44}(1,4,5)$	180	200	*180*	53.72
$C_{44}(1,4,7,10)$	632	830	[648]	$>$ 86400.00
$C_{46}(1,4)$	46	50	46	0.07
$C_{46}(1,5,8)$	296	374	**294**	1104.35

show the following data: the name of the instance, the number of crossings produced by the best and worst heuristics of [1], respectively, the optimal number of crossings (when successfully computed by our approach), and the runtime of our algorithm. However, as some instance are far too large for exact solution, we had to set a general time limit of 24 hours. Whenever this limit was reached, we report the best crossing number found instead of the optimal solution; the figures are then put into brackets. Where an optimal solution was found for an instance that was not solved to proven optimality before, we use italics. Bold figures indicate that our algorithm could improve the best heuristic solution.

It is remarkable that many instances can be solved very quickly by our approach while others cannot even be solved in one CPU day. In other words, the border line between easy instances (those solvable within 25 seconds, say) and hard ones (those unsolved even in one day) is very sharp, few instances do not fall into one of these categories.

Our results can also help to evaluate the quality of the heuristic methods. In fact, it turns out that many heuristics proposed by [1] are able to find optimal or near-optimal solutions even for larger instances. In summary, we think that small to medium sized instances should be solved to optimality in general, whereas for larger instances one can at least be confident that the heuristic solution is not too far away from the optimum.

The algorithm we used for solving the MAXCUT problem is generally better adapted to sparse graphs. This is reflected in the runtime figures presented in this section. Therefore, practical instances tend to be easy for our approach.

5 Conclusion and Future Work

We have presented a new exact algorithm for the fixed linear crossing number problem, running significantly faster than earlier exact algorithms. The essential part of our approach is the reduction to the maximum cut problem. After this transformation, the problem can be solved with a sophisticated mathematical programming algorithm, based on the extensive knowledge that has been gathered for the maximum cut problem by intensive research. Moreover, testing the existence of a planar fixed linear embedding of a given graph can be done in an easy way using this transformation. We believe that this principle can also be applied to other linear embedding problems with different objective functions.

Our experimental results show that many medium sized instances can be solved very quickly by our approach. However, for many large instances we cannot find optimal solutions. For these instances, the heuristics proposed by Cimikowski [1] are a good compromise between running time and quality. In fact, our evaluation shows that in most cases at least one of these heuristics is able to find the optimal solution.

In consequence, we plan to integrate good heuristics into our branch-and-cut algorithm in order to further improve running times. In general, this can be done in the same way as in the branch-and-bound approach. We are convinced that this will considerably increase the performance of our approach.

Since the generation of edge crossings largely depends on the original vertex ordering, it is crucial to study the general version of the linear crossing number problem. We plan to develop heuristic or exact algorithms finding vertex orderings leading to a minimal number of potential edge crossings. Having done this, we will be able to evaluate our approach on instances without a predetermined order of vertices. In particular, we plan to test its performance on the graphs in the well-known Rome library. As these graphs are usually very sparse, we are convinced that we will be able to solve most of these instances to optimality.

Acknowledgement. We would like to thank Frauke Liers for providing us her implementation of a branch-and-cut algorithm for the maximum cut problem. Moreover, we are grateful to Robert Cimikowski for making his implementation and experimental data available to us.

References

1. R. Cimikowski. Algorithms for the fixed linear crossing number problem. *Disc. Appl. Math.*, 122:93–115, 2002.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, 4:435–282, 1994.
3. Z. Galil, R. Kannan, and E. Szemerédi. On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines. *J. Comput. System Sci.*, 38(1):134–149, 1989.
4. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Alg. Disc. Meth.*, 4:312–316, 1983.
5. R. S. Gilbert and W. K. Kleinöder. CNMgraf – graphic presentation services for network management. In *Proc. 9th Symposium on Data Communication*, pages 199–206, 1985.
6. T. Harju and L. Ilie. Forbidden subsequences and permutations sortable on two parallel stacks. In *Where mathematics, computer science, linguistics and biology meet*, pages 267–275. Kluwer, 2001.
7. M. Laurent. The max-cut problem. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliography in Combinatorial Optimization*. Wiley, 1997.
8. M. Laurent and F. Rendl. Semidefinite programming and integer programming. In *Discrete Optimization*, pages 393–514. Elsevier, 2005.
9. F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. Computing exact ground states of hard Ising spin glass problems by branch-and-cut. In *New Optimization Algorithms in Physics*, pages 47–69. Wiley-VCH, 2004.
10. S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Trans. Comput.*, 39(1):124–127, 1990.
11. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001.
12. T. A. J. Nicholson. Permutation procedure for minimizing the number of crossings in a network. In *Proc. IEEE*, volume 115, pages 21–26, 1968.
13. A. L. Rosenberg. DIOGENES, circa 1986. In *Proc. VLSI Algorithms and Architectures*, volume 227 of *LNCS*, pages 96–107. Springer, 1986.