An Experimental Study of Crossing Minimization Heuristics

Carsten Gutwenger[1] and Petra Mutzel[2]

[1] Stiftung caesar
Ludwig-Erhard-Allee 2, D-53175 Bonn, Germany
[2] Vienna University of Technology
Favoritenstr. 9–11 E186, A-1040 Vienna, Austria

Abstract. We present an extensive experimental study of heuristics for crossing minimization. The heuristics are based on the planarization approach, so far the most successful framework for crossing minimization. We study the effects of various methods for computing a maximal planar subgraph and for edge re-insertion including post-processing and randomization.

1 Introduction

The crossing minimization problem is one of the crucial problems in graph drawing (see, e.g., [22]). Here, the task is to find a drawing of a graph in the plane with the minimum number of crossings. The crossing number represents a fundamental measure of non-planarity of graphs and has been studied for more than 40 years by graph theorists. The algorithmical problem of computing the crossing number has also been studied in the context of VLSI-layout. So far, only a few infinite classes of graphs exist, for which the crossing number is known. We do not even know the asymptotic value for the complete graph K_n with n vertices and for the complete bipartite graph $K_{n,n}$ with $2n$ vertices, as n tends to infinity [23].

We do know, however, that the crossing number problem and several of its variants are NP-hard [13,3]. While many other prominent NP-hard problems have been successfully attacked with integer programming and branch-and-bound techniques, no similar approach to the crossing number problem is known to date. To our knowledge, no exact algorithm exists that is able to solve even small instances of the crossing number problem to provable optimality within reasonable computation time. For graphs with bounded degrees, Even et al. [8] have recently suggested an approximation algorithm in which the sum of the numbers of vertices and crossings is $O(\log^3 |V|)$ times the minimum sum thus improving the results of $O(\log^4 |V|)$ by Bhatt and Leighton [2] and Leighton and Rao [17]. Grohe [9] has given an exact algorithm that works in quadratic time if the crossing number is fixed. Both algorithms are rather of theoretical nature and have so far not been useful for solving practical instances. Much easier to tackle is the bipartite crossing number, for which a vast amount of experimental papers exist in the literature (see, e.g., [16,7]). Experimental papers in other fields

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 13–24, 2004.

include, e.g., a collection of experimental studies in graph drawing by Vismara et al. [24] and the paper by Brandenburg et al. on force-directed methods [4].

The only experimental state-of-the-art study – to our knowledge – concerning crossings for general graphs has been published in [6]. It includes four different algorithms for orthogonal graph drawing on the benchmark set described above. Two of these algorithms were based on the topology-shape-metrics approach, and the others are incremental algorithms that focus on a small area and a small number of bends. The results show that the former two algorithms are superior in terms of the criteria number of crossings, number of bends, area, and edge length. E.g., the best algorithm had up to 8 times fewer crossings. Therefore, we decided to solve the crossing minimization problem heuristically using a 2-step planarization approach, which we will discuss in Section 3. There, we will also describe our strategies (some of them are new) for crossing reduction. Section 4 contains a discussion of our extensive experimental results. Our paper ends with a section on conclusions (see Section 5). Before we can start the technical part of the paper, we need to introduce some basic mathematical terms (see Section 2).

2 Preliminaries

In a drawing of a graph $G = (V, E)$ each vertex $v \in V$ is mapped to a distinct point p_v in the plane and each edge $(u, v) \in E$ is mapped to a closed simple curve that connects the points p_u and p_v and does not pass through the image of any other vertex. If two curves share an interior point p, we say that they cross at p. The crossing number $cr(G)$ is the minimal number of crossings in any drawing of G. The crossing number problem is the problem of finding the crossing number for a given graph G.

The graphs that can be drawn without any edge crossings are called *planar graphs*. A planar drawing of a graph divides the plane into regions called *faces*. Every drawing defines a *planar* and a *combinatorial embedding* of the graph G. Such an *embedding* essentially fixes the topology of the graph. A *combinatorial embedding* is defined as a clockwise ordered list of adjacent neighbors for each vertex $v \in V$. When, in addition, the outer face is fixed, the combinatorial embedding is also called a *planar embedding* of G. An alternative definition of a combinatorial embedding is for each face f an anti-clockwise ordered list of the edges bordering f. Given a planar graph, a combinatorial embedding can be computed in linear time [5,19]. In general, a planar graph can have an exponential number of combinatorial embeddings. In the following section, we will use the name *embedding* for planar or combinatorial embedding.

3 The Planarization Approach

In practice, the crossing minimization problem is solved heuristically using a 2-step planarization approach. In a first step, a small number of edges is deleted from $G = (V, E)$ in order to obtain a planar graph P. In a second step, the

edges are re-inserted into the planar graph P while trying to keep the number of crossings small.

3.1 Methods for Computing the Planar Subgraph

For the first step, we need to solve the so-called *maximum planar subgraph problem*, which has been shown to be NP-hard [18]. If the number of edges to be deleted is small, the exact branch-and-cut algorithm suggested in [15] is able to provide a provably optimal solution quite fast. However, the method is quite complicated to understand and to implement: Moreover, if the number of deleted edges exceeds 10, the algorithm usually needs far too long to be acceptable for practical computation. Since we are interested in approaches for practitioners, we did not involve this exact method into our studies. Interested readers are refered to the study of Ziegler [25] concerning the number of deleted edges in the *Rome library* benchmark set.

A widely used standard heuristic for finding a maximal planar subgraph is to start with the empty graph, and to iteratively try to add the edges one by one. In every step, a planarity testing algorithm is called for the obtained graph. If the addition of an edge would lead to a non-planar graph, then the edge is disregarded; otherwise, the edge is added permanently to the planar graph obtained so far. After $|E|$ iterations (planarity tests), we have obtained a maximal planar subgraph P of G, i.e., a subgraph of G which will get non-planar as soon as any of the edges in $G - P$ will be added. We will denote this method as MAXIMAL. The standard (and also our) implementation needs a running time of $O(|E||V|)$. Theoretically, this can be improved to nearly linear running time using so-called online-planarity testing algorithms (e.g., [1,21]), but we are not aware of any correct implementation.

An alternative to this method is to use the planarization algorithm based on PQ-trees suggested in [12,14]. Observe, that this method cannot guarantee to derive a maximal planar subgraph. The theoretical worst case running time is $O(|V|^2)$. In practice it is much faster.

The quality of the results can be improved by introducing random events and calling them several times. The PQ-tree based algorithm starts by computing a so-called st-numbering. Our random event was simply to choose a random edge of E in order to get s and t. We studied the effects of up to 100 calls. We denote these methods as PQ1, PQ10, PQ50, and PQ100 for 1, 10, 50, and 100 iterations.

3.2 Edge Re-insertion Strategies

Fixed Embedding. Also the edge re-insertion step is a NP-hard optimization problem [25]. The standard algorithm used in practice re-inserts the edges $e_1, e_2, \ldots, e_k$ iteratively starting with a given planar embedding of G. The approach is based on the observation that an edge e_i crosses an edge in P if and only if it uses an edge in the geometric dual graph of P. Hence, the problem of re-inserting only one edge into P with a given planar embedding can be solved via

a simple shortest-path computation in the extended dual graph of P. (We need to extend the dual graph in order to connect the end-vertices of e_i with the dual graph.) After each insertion step i, the crossings generated by edge e_i are substituted by artificial vertices so that the resulting graph $G \cup \{e_1, \ldots, e_i\}$ becomes planar again ($i = 1, \ldots, k$). The theoretical worst case running time for inserting k edges of our implementation is $O(\sum_{i=1}^{k}(|V| + \sum_{j=0}^{i-1} c_j)) = O(k(|V| + |C|))$, where c_j is the number of crossings introduced in step j, $c_0 = 0$, and C the number of crossings in the final drawing. In practice, it is much faster, since the update of the dual graphs are implemented efficiently. We denote this re-insertion method as FIX.

Variable Embedding. However, the quality of the resulting drawing highly depends on the chosen embedding for P. In [11], Gutwenger et al. have given a linear time algorithm based on the SPQR-tree data structure for inserting one edge into a planar graph P so that the number of crossings in $P \cup \{e\}$ over the set of all possible planar embeddings of P is minimized. Our implementation has the same theoretical running time as the variant FIX. We denote this re-insertion method as VAR.

Constrained Crossing Minimization. Obviously, re-insertion of all edges at the same time will improve the solution. However, no practically efficient algorithm is known. The *constrained crossing minimization problem* asks for the minimum number of crossings obtained by a set of edges F when inserted into a planar graph P, while the embedding of P is not changed. The problem has been investigated in [20,25]. Experiments show that it can only be solved to provable optimality if there are less than 10 re-inserted edges — and even then, the running time is relatively high. Therefore, we did not include this method into our experiments.

Post-Processing Strategies. The idea of the post-processing strategies is to iteratively delete an edge from the drawing and to re-insert it again. Our strategies vary in the set and/or number of edges involved in the deletion and re-insertion process, and the order of processing them.

The variant INS involves exactly those edges which have been deleted already in the planar subgraph step, whereas the variants ALL and MOST involve the whole set of edges E in the original graph G. An iteration takes either the whole set (in variant INS and ALL) or $x\%$ of this edges (variant MOST $x\%$) iteratively (one after the other). The procedure stops only if within one iteration no improvement has been made. How do we choose the edges in variant MOST $x\%$? After each iteration, we sort the involved edge set in descending order according to the number of crossings they are involved in. Now, only the first $x\%$ edges of this list are taken for re-insertion. The variant in which there is no post-processing routine is called NONE.

The post-processing procedure can be implemented efficiently by updating the dual graph only at those regions, in which changes did occur. We did this for our FIX strategy. In principal, such an update is also possible for the VARIABLE embedding setting [1]. However, we are not aware of any implementation of this

algorithm. This explains the big running time discrepancy in the post-processing procedure between the FIXED and VARIABLE embedding setting.

Permutations. After a whole deletion and re-insertion process of the chosen strategy for embedding FIX/VAR and a strategy for post-processing NONE/INS/ ALL/MOST, we get a certain crossing number. Our permutation variant does nothing else, but repeating the whole edge re-insertion process and keeping the best results. The random effect exists in choosing a different ordering of the edges in $G - P$ for the initial re-insertion step. The notation PERMi gives the number of these repetition rounds. We have experimented with PERM1, PERM2, PERM10, and PERM20.

4 Experimental Results

For our computational experiments, we have used our new graph drawing library which is the basis of the GoVisual layout tools [10]. Our computational experiments ran on a PC with Pentium 4, 2.4 GHz, 512 MB RAM, under MS Windows 2000, and our C++-code has been compiled with MS Visual C++.NET (Visual C++ 7). We used the benchmark set which is commonly known as the *Rome library*. It contains 11.528 graphs[1] with 10 to 100 vertices, and has been generated from a core set of 112 graphs used in *real-life* software engineering and database applications. The library is available via `http://www.dia.uniroma3.it/people/gdb/ wp12/undirected-1.tar.gz`.

4.1 Results of the Planar Subgraph Computations

In our first experiments, we measured the average number of deleted edges of the five strategies for computing a planar subgraph for each group with vertex size i, $i = 10, \ldots, 100$, separately. The results improve significantly as the number of permutations increases. While the number of deleted edges went up to 19 (on average) for PQ1, it was about 16 for PQ10, and 15 for PQ50 and PQ100. It seems that it really pays off to run the planar subgraph algorithm many times. However, the results for 50 and 100 permutations are very close to each other. This effect comes from the random effect chosen for our implementation (see Section 3.1). By choosing other randomization techniques, this effect may occur at a higher number of permutations. Since the running time of our implementation is relatively small, it seems that taking 100 iterations of the PQ-based algorithm (PQ100) is a good choice. The running time for PQ1 was below 0.002 seconds, while the time for MAXIMAL increased to 0.022 seconds at 100 vertices. The time for PQ100 was about 0.34 seconds for instances with 100 vertices.

4.2 Results for the Edge Re-insertion Step

FIX vs. VAR. Figure 1 shows the number of crossings generated with the variants PQ1 vs. PQ100 and FIX vs. VAR. The VAR strategy seems to get

[1] originally, it were 11,582 graphs, but some of the files are corrupted

better results than the FIX strategy. This effect is outperformed, however, by starting with a better planar subgraph for edge re-insertion (i.e., taking PQ100 instead PQ1). Because of this, we decided to stay with PQ100 for our further experiments.

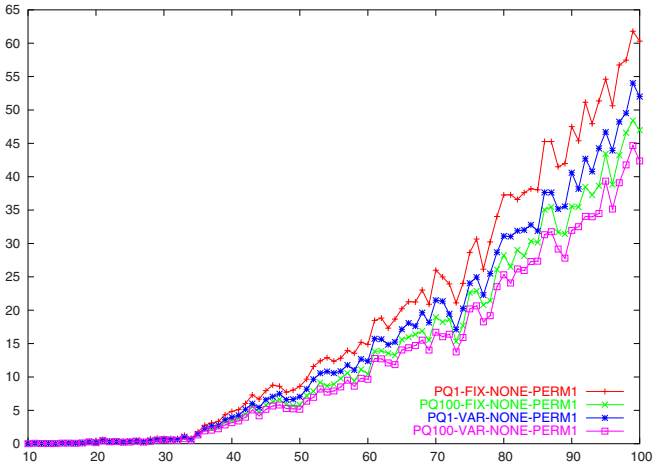

Fig. 1. The effect of the edge re-insertion strategies FIX and VAR.

On the instances with 100 vertices we get 60.32 crossings on average. This is about the same number which was also reported in the study by Di Battista et al. [6]. By computing a better planar subgraph the number of crossings reduces from 60.32 to 46.97 (about 22% improvement). This number can further be reduced to 42.37 by choosing the best embedding for each inserted edge. This is already an improvement of about 30%.

For the remaining Figures, we decided to show the relative improvement of the variants acording to the *standard method* PQ1-FIX-NONE-PERM1, which was also used in [6].

Post-Processing. Figure 2 shows the effect of the post processing variants for the VAR strategy. Taking the inserted edges as candidates for re-insertion is already much better than the NONE strategy (no post-processing at all). However, the results can be improved a lot more by taking the whole set of edges into account. We observe that already taking into account 25% of the edges gives similar results to the options MOST100 and ALL. ALL, MOST100, and MOST25 improve the number of crossings up to 46.11% (corresponds to 32.5 crossings at 100 vertices) compared to the standard method. For the FIXED strategy, the picture looks very similar. Here, the improvement coming from post-processing in comparison to NONE is a little bit better.

We also measured the running times of the post processing variants. Strategies INS and MOST10 have about the same running time (up to 0.18 seconds), which is up to a factor of 2.5 slower than the NONE variant. MOST25 needs

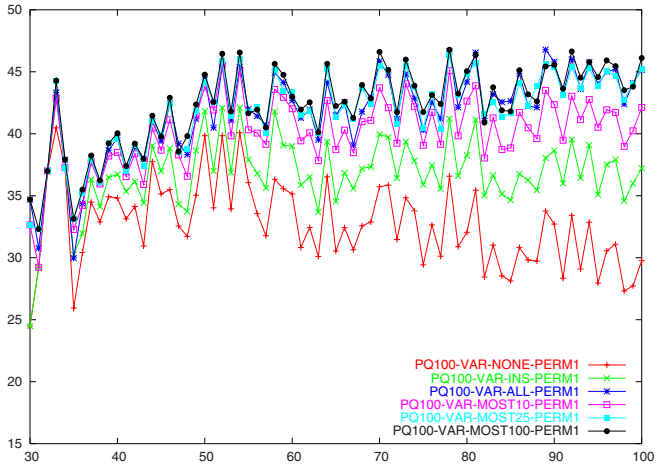

Fig. 2. The effect of the post processing variants compared to the standard method.

up to 0.35 seconds. Not surprisingly, MOST100 and ALL have a similar running time (up to 0.53 seconds), and are the slowest variants. For the FIX strategy, the running time of all versions is about the same. The time for variant PQ100-FIX-MOST100-PERM1 is only 0.01 seconds larger than the time for variant PQ100-FIX-NONE-PERM1. This comes from the efficient update operations mentioned in Section 3.2.

Permutations. Figure 3 shows the effect of the permutation variants for FIX. As expected, the results improve with a higher permutation number. However, we observe that the post-processing effects are stronger than the PERM20 effects: all four strategies PQ100-FIX-MOST100-PERMi lead to much better results (between 41.77% and 50.25% improvement at 100 vertex instances) than the four strategies PQ100-FIX-NONE-PERMi ($i = 1, 2, 10, 20$) (between 22.12% and 32.36%).

Obviously, the running time increases a lot: PERM20 needs about 20 times the running time of PERM1 (since the edge re-insertion step dominates the procedure). For VAR, the situation is similar. The only difference is that the upper three curves are closer to the others.

PQ1 vs. PQ100. Finally, we have tested the effects of the maximal planar subgraphs heuristics PQ1 and PQ100 again for the best edge re-insertion strategies. Figure 4 shows that PQ100-FIX-ALL-PERM20 gives an improvement of 49.55% compared to the standard method, whereas PQ1-FIX-ALL-PERM20 gives an improvement of 47.22%. The running times for both variants are about the same. Interestingly, this is not true for the strategies PQ100-VAR-ALL-PERM1 and PQ1-VAR-ALL-PERM1. Due to the better planar subgraph achieved with PQ100-VAR-ALL-PERM1, the running times differ by 0,21 seconds at the instances with 100 vertices (0.48 compared to 0.69). Surprisingly, the PQ100 strat-

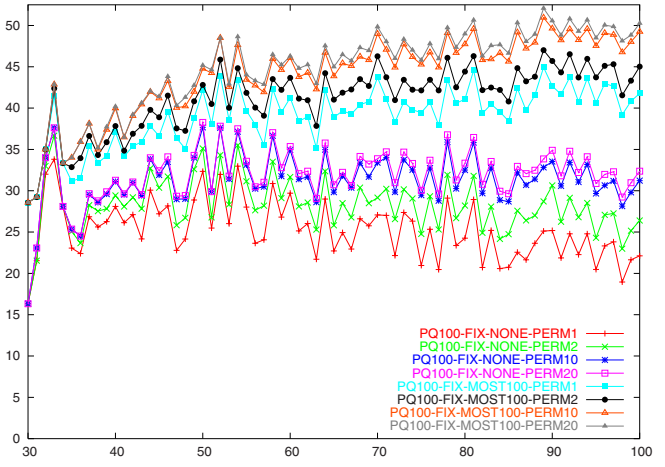

Fig. 3. The effect of the PERM variants.

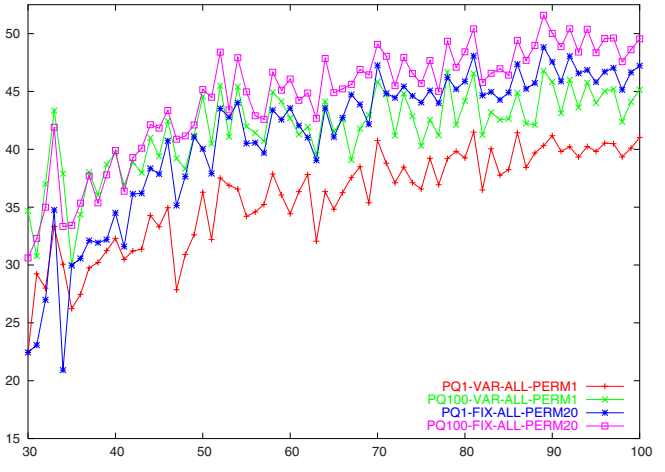

Fig. 4. The relative effect of the maximal planar subgraph variants.

egy is faster than the PQ1 strategy. The relative improvements in the number of crossings are 45.15% and 41.03%, respectively.

Conclusions. We have run all possible experiments with strategies FIX and VAR, the different post-processing strategies NONE and ALL, and the permutation numbers 1 and 10. Figure 5 shows the results for these eight strategies. The best results gives PQ100-VAR-ALL-PERM10 (51.63% improvement at 100 vertex instances corresponding to 29.18 crossings) followed by PQ100-FIX-ALL-PERM10 (48.42% corresponding to 31.11 crossings). The three strategies us-

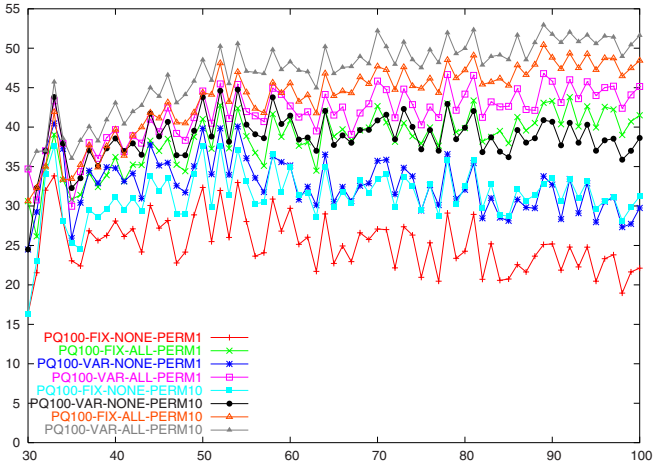

Fig. 5. The results for a collection of strategies.

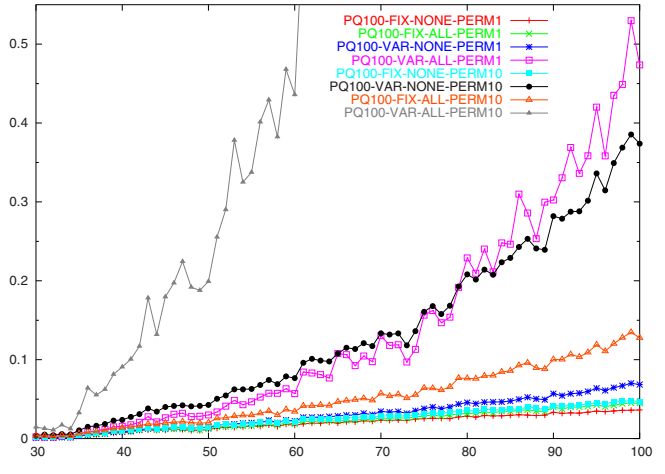

Fig. 6. The running times for the results displayed in Figure 5.

ing post-processing ALL are the four *winners* in this Figure. Interestingly, the best method without any post-processing, PQ100-VAR-NONE-PERM10, is in competition with the strategy PQ100-FIX-ALL-PERM1 (with up to 45% improvement). This means that, in order to be competitive with post-processing, 10 permutations are not enough. Additional help is needed, here in form of the strategy VAR compared to FIX.

Figure 6 shows the running times for the results displayed in Figure 5. We observe that the running time of PQ100-VAR-NONE-PERM10 is much slower than the running time of PQ100-FIX-ALL-PERM1. Hence PQ100-FIX-ALL-PERM1

is the clear winner among these two strategies. The best strategy displayed in Figure 6, PQ100-VAR-ALL-PERM10, has the highest running time. Interestingly, the second best strategy, PQ100-FIX-ALL-PERM10, has the fifth best running time (up to 0,14 seconds).

Table 1. The ranking list of the *best* heuristics.

Rank	Crossings	Time (sec)	MPS	FIX / VAR	POST	PERM
1	28.56	8.778	PQ100	VAR	ALL	PERM20
2	28.61	8.563	PQ100	VAR	MOST100	PERM20
3	28.66	5.902	PQ100	VAR	MOST25	PERM20
4	29.09	4.359	PQ100	VAR	MOST100	PERM10
5	29.35	3.060	PQ100	VAR	MOST25	PERM10
6	30.01	0.259	PQ100	FIX	MOST100	PERM20
7	30.43	0.258	PQ100	FIX	ALL	PERM20
8	30.62	0.130	PQ100	FIX	MOST100	PERM10
9	31.11	0.128	PQ100	FIX	ALL	PERM10
10	31.64	0.112	PQ100	FIX	MOST25	PERM10
11	33.16	0.054	PQ100	FIX	MOST100	PERM2
12	33.29	0.053	PQ100	FIX	ALL	PERM2
13	34.14	0.050	PQ100	FIX	MOST25	PERM2
14	35.12	0.046	PQ100	FIX	MOST100	PERM1
15	35.29	0.045	PQ100	FIX	ALL	PERM1
16	36.09	0.043	PQ100	FIX	MOST25	PERM1
17	38.93	0.040	PQ100	FIX	MOST10	PERM1
18	46.98	0.036	PQ100	FIX	NONE	PERM1
19	60.32	0.002	PQ1	FIX	NONE	PERM1

Table 1 shows a ranking list of the best obtained results in the following sense: We have sorted all strategies according to their average crossing number achieved at the instances with 100 vertices. Running through this list, we deleted all those strategies, which have obtained worse crossing number and worse running time than another strategy on the list. It is interesting that the best five strategies are based on VAR, and all of the remaining strategies in our ranking list are based on FIX. This stems from the fact, that from a certain point on, the VAR strategy takes much more time than the FIX strategy. E.g., the strategy PQ100-VAR-ALL-PERM1 has been eliminated from the table through PQ100-FIX-MOST100-PERM20. It seems that the effect coming from the permutations is stronger than the positive VAR effect, at least until a certain point.

Figure 7 shows a selection of the best obtained results listed in Table 1. The winners are PQ100-VAR-ALL-PERM20 (with improvement of 52.65% corresponding to 28.56 crossings) and PQ100-VAR-MOST25-PERM20, followed by PQ100-VAR-MOST100-PERM10 and PQ 100-VAR-MOST25-PERM10. Only then, the FIX strategies appear on the list.

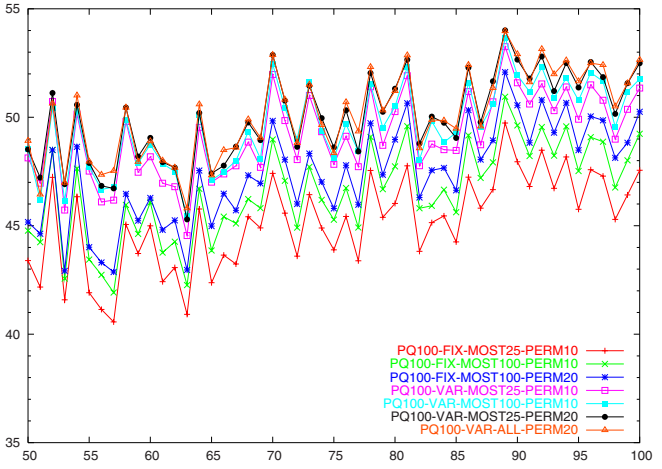

Fig. 7. The best obtained results within our study.

5 Conclusions

We have conducted extensive experimental studies on the crossing minimization problem for a benchmark set of graphs. The main conclusions are:

1. Post-processing always helps. It is recommended not to restrict the post-processing procedure to the inserted edges. Already re-inserting 25% of all the edges helps a lot.
2. Permutations and random effects help, but not as well as post-processing.
3. It is important to start with a good planar subgraph. A better subgraph leads not only to an improved number of achieved crossings, but also to an improved running time of the algorithm.
4. The re-insertion within a variable embedding setting is still worth doing, even if post-processing is used.

Acknowledgement. We thank Sebastian Leipert for providing his implementation of the PQ-based planarity testing, planarity embedding, and planar subgraph algorithms.

References

[1] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.
[2] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984.
[3] D. Bienstock. Some provably hard crossing number problems. *Discrete & Computational Geometry*, 6:443–459, 1991.

[4] F.J. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In F.J. Brandenburg, editor, *GD '95*, volume 1027 of *LNCS*, pages 76–87. Springer-Verlag, 1996.
[5] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Comput. Syst. Sci.*, 30(1):54–76, 1985.
[6] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7:303–326, 1997.
[7] T. Eschbach, W. Günther, R. Drechsler, and B. Becker. Crossing reduction by windows optimization. In M.T. Goodrich and S.G. Kobourov, editors, *Graph Drawing '02*, volume 2528 of *LNCS*, pages 285–294. Springer-Verlag, 2002.
[8] G. Even, S. Guha, and B. Schieber. Improved approximations of crossings in graph drawing and VLSI layout area. In *Proc. 32nd ACM Symp. Theory of Comp. (STOC'00)*, pages 296–305. ACM Press, 2000.
[9] M. Grohe. Computing crossing numbers in quadratic time. In *Proc. 32nd ACM Symp. Theory of Computing (STOC'00)*, pages 231–236. ACM Press, 2000.
[10] C. Gutwenger, K. Klein, J. Kupke, S. Leipert, M. Jünger, and P. Mutzel. Govisual software tools. http://www.oreas.de, 2002.
[11] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *Proc. Ninth Ann. ACM-SIAM Symp. Discr. Algorithms (SODA '2001)*, pages 246–255, Washington, DC, 2001. ACM Press.
[12] R. Jayakumar, K. Thulasiraman, and M. N. S. Swamy. $O(n^2)$ algorithms for graph planarization. *IEEE Trans. on Computer-Aided Design*, 8:257–267, 1989.
[13] M. R. Johnson and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.
[14] M. Jünger, S. Leipert, and P. Mutzel. A note on computing a maximal planar subgraph using PQ-trees. *IEEE Trans. Computer-Aided Design*, 17(7), 1998.
[15] M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16(1):33–59, 1996.
[16] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Gr. Alg. & Appl. (JGAA)*, 1(1):1–25, 1997.
[17] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.
[18] P.C. Liu and R.C. Geldmacher. On the deletion of nonplanar edges of a graph. In *10th. S-E Conf. Comb., Graph Theory, and Comp.*, pages 727–738, 1977.
[19] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.
[20] P. Mutzel and T. Ziegler. The constrained crossing min. problem. In J. Kratochvíl, editor, *GD '99*, volume 1731 of *LNCS*, pages 175–185. Springer-Verlag, 1999.
[21] J. A. La Poutré. Alpha-algorithms for incremental planarity testing. In *Proc. 26th Annual ACM Symp. Theory of Computation (STOC)*, pages 706–715, 1994.
[22] H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Graph Drawing '97*, volume 1353 of *LNCS*, pages 248–261. Springer-Verlag, 1997.
[23] R.B. Richter and C. Thomassen. Relations between crossing numbers of complete and complete bipartite graphs. *Amer. Math. Monthly*, pages 131–137, 1997.
[24] L. Vismara, G. Di Battista, A. Garg, G. Liotta, R. Tamassia, and F. Vargiu. Experimental studies on graph drawing algorithms. *Software – Practice and Experience*, 30:1235–1284, 2000.
[25] T. Ziegler. *Crossing Minimization in Automatic Graph Drawing*. PhD thesis, Max-Planck-Institut für Informatik, Saarbrücken, 2000.