Alcom-FT Technical Report Series
ALCOMFT-TR-01-148

Removing Cycles for Minimizing Crossings*

Camil Demetrescu
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
demetres@dis.uniroma1.it

Irene Finocchi
Dipartimento di Scienze dell'Informazione
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
finocchi@dsi.uniroma1.it

Abstract

We consider the one-sided crossing minimization problem (CP): given a bipartite graph G and a permutation x_0 of the vertices on a layer, find a permutation x_1 of the vertices on the other layer which minimizes the number of edge crossings in any straightline drawing of G where vertices are placed on two parallel lines and sorted according to x_0 and x_1. Solving CP represents a fundamental step in the construction of aesthetically pleasing layouts of hierarchies and directed graphs, but unfortunately this problem has been proved to be NP-complete.

We first address the strong relation between CP and the problem of computing minimum feedback arc sets in directed graphs and we devise a new approximation algorithm for CP, called PM, that exploits this dependency. Then, we experimentally and visually compare the performance of PM with the performance of well-known algorithms and of recent attractive strategies. Experiments are carried out on different families of randomly generated graphs, on pathological instances for CP, and on real test sets. Performance indicators include both number of edge crossings and running time, as well as structural measures of the problem instances.

We found CP to be a very interesting and rich problem from a combinatorial point of view. Our results clearly separate the behavior of the algorithms, proving the effectiveness of PM on most test sets and showing tradeoffs between quality of the solutions and running time. However, if the visual complexity of the drawings is considered, we found no clear winner. This confirms the importance of optimizing also other aesthetic criteria such as symmetry, edge length, and angular resolution.

1 Introduction

Directed graphs are frequently used to model complex systems in a variety of fields. A limited list of examples includes procedure call dependencies and class hierarchies in software engineering, PERT diagrams in project planning, is-a relationships in knowledge representation, and several structures from economic and social sciences and from graphic user interfaces. A common practice to understand the structure of the systems consists of using visualizations of these graphs. Obviously, the viewer can effectively benefit from such drawings only if they are clear and readable; for this reason many different aesthetic criteria have been proposed and studied in the graph drawing field. In particular, two of them appear to have a great impact on the readability of layouts of directed graphs [4, 28, 31]: edges should point in the same direction and crossings between edges should be avoided. Actually, these criteria make it easier for the viewer to trace paths in the graph.

*This work has been partially supported by the IST Programme of the EU under contract n. IST-1999-14.186 (ALCOM-FT), by the Italian Ministry of University and Scientific Research (Project "Algorithms for Large Data Sets: Science and Engineering".

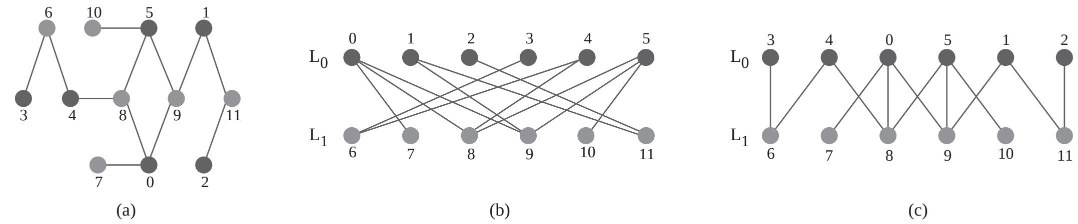


Figure 1: (a) A bipartite graph G; (b) a bipartite drawing of G; (c) a bipartite drawing of G with the minimum number of edge crossings keeping L_1 fixed.

Reducing crossings and edges "against the flow" were the guidelines followed in the design of a well-known method for drawing acyclic directed graphs, usually denoted *hierarchical approach* [4, 31]. This method aims at creating polyline layered drawings, where vertices are constrained to lie on a set of parallel lines called *layers*, and consists of three main steps: (1) assigning the vertices of the digraph with layers; (2) ordering the vertices within each layer so as to minimize the number of edge crossings; (3) choosing vertical and horizontal coordinates for vertices and edge bends. If the digraph at hand is not acyclic, a preprocessing step is also performed by the algorithm, where the direction of some edges is reversed in order to eliminate all the directed cycles.

According to the hierarchical approach, the number of crossings is usually reduced by employing a *layer-by-layer sweep* heuristic [11] which requires solving the following problem: given a two-layered graph and a permutation of the vertices on a layer, find a permutation of the vertices on the other layer which minimizes the number of edge crossings. In the following we refer to this problem as the *one-sided crossing minimization problem* (CP).

More formally, let $G = (L_0, L_1, E)$ be a bipartite graph, with vertex set $L_0 \cup L_1$ and edge set $E \subseteq L_0 \times L_1$. Let x_0 and x_1 be two permutations of the vertices in L_0 and L_1, respectively. Such permutations induce a family of *bipartite drawings* of G, i.e., drawings in which vertices lie on two parallel lines, sorted according to x_0 and x_1, and edges are represented as straight lines (see also Figure 1). All the bipartite drawings in the family have the same number of edge crossings, which depends only on permutations x_0 and x_1 and not on the actual positions of vertices: we call this number $cross(G, x_0, x_1)$. Moreover, for any fixed x_1, let $opt(G, x_1)$ be the minimum number of edge crossings in a bipartite drawing of G with layer L_1 ordered according to x_1, i.e., $opt(G, x_1) = min_{x_0} cross(G, x_0, x_1)$. Based on this notation, the one-sided crossing problem can be formally stated as follows:

CP: Given a bipartite graph $G = (L_0, L_1, E)$ and an ordering x_1 of the vertices in L_1, find an ordering x_0 of the vertices in L_0 such that $cross(G, x_0, x_1) = opt(G, x_1)$.

The one-sided crossing problem has been proved to be NP-complete in [17] and extensive research has been devoted since the early 80's to the design and analysis of heuristics and approximation algorithms for it [5, 10, 13, 17, 19, 22, 24, 25, 26, 31, 32, 34]. The study in [22] also presents a branch-and-cut algorithm for computing exact optima and assesses an experimental comparison of well-known methods for solving CP on randomly generated bipartite graphs. The outcome of this study shows a considerable gap between the theoretical results and the real behavior of the most popular algorithms for CP even on very small instances with 20 vertices per layer: namely, algorithms achieving constant approximation ratios are often outperformed by algorithms with worse theoretical performances or even by simple heuristics. This gap, together with the impossibility of computing true optima except for graphs with at most 60 vertices in L_0, led researchers to reconsider the one-sided crossing problem from different perspectives, thus yielding to the design of some recent new algorithms for it [10, 25, 34]. Among them, the algorithm presented in [34] stands out from a theoretical point of view, obtaining the best approximation ratio known for CP.

In this paper we first address the strong relation between CP and the problem of computing minimum feedback arc sets in directed graphs suggested by Sugiyama, Tagawa, and Toda [31]: we design a relaxed penalty minimization scheme (in short, RPMS) that exploits this dependency and we devise a polynomial time approximation algorithm for CP, called PM. Then, we experimentally and visually compare the performance of PM with the performance of algorithms well-known in literature and of recent attractive strategies. Experiments are carried out on different families of randomly generated graphs, on pathological instances for CP, and on real test sets. Performance indicators include both number of edge crossings and running time, as well as structural measures of the problem instances.

We found the one-sided crossing problem to be very interesting and rich from a combinatorial point of view. Our results clearly separate the behavior of the algorithms both on synthetic and on real test sets, proving the effectiveness of PM on most instances, highlighting which algorithms are more robust, and showing tradeoffs between quality of the solutions and running time. However, if the visual complexity of the drawings is considered, we found no clear winner: as a matter of fact, all the algorithms find solutions whose quality is not far from the number of crossings which are combinatorially unavoidable. This number is a lower bound to the optimum and may even grow exponentially with the edge density of the bipartite graph, thus representing a strong barrier for obtaining aesthetically pleasing layouts of dense graphs. However, besides the fact that graphs arising in most real applications are very sparse, the visual comparison between the drawings produced by whatever algorithm for CP and layouts obtained with no optimization attempts clearly proves that trying to minimize edge crossings is very useful even on dense instances. We also observed that drawings with more crossings may appear more readable simply because of other aesthetic criteria, such as symmetry, edge length, and angular resolution. For this reason, we believe that variants of the one-sided crossings minimization problem taking into account also these criteria should be studied.

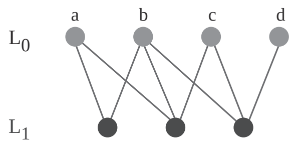
The remainder of this paper is organized as follows. Section 2 introduces the relaxed penalty minimization scheme and presents algorithm PM. Implementation details concerning PM are discussed in Section 3 and Section 4 is devoted to the description of the experimental framework. Finally, Section 5 discusses the computational results obtained from the experimentation and Section 6 addresses concluding remarks.

2 The penalty minimization approach

In this section we discuss theoretical issues concerning the penalty minimization approach for solving the one-sided crossing problem. In particular, in Section 2.1 we prove that any approximation algorithm for the *minimum feedback arc set problem* (FAS) can be used for approximating CP by the same ratio. This reduction leads to the design of the relaxed penalty minimization scheme described in Section 2.2. An application of RPMS is finally provided in Section 2.3, where a new approximation algorithm for CP is presented.

Before stating our results, we introduce some preliminary crossing theory, making use of standard graph drawing terminology from [11]. Let $G = (L_0, L_1, E)$ be a bipartite graph and let x_1 be an ordering of the vertices in L_1. In the rest of the paper we assume that L_1 is the fixed layer of G. Moreover, we denote with n_0, n_1, and m the cardinality of L_0, L_1, and E, respectively, and with n the total number of vertices $n_0 + n_1$.

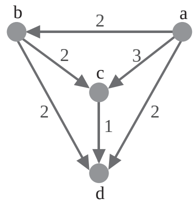
For any two vertices $u, v \in L_0$, the number of crossings between edges incident to u and edges



(a)

	a	b	c	d
a	–	1	0	0
b	3	–	1	0
c	3	3	–	0
d	2	2	1	–

(b)



(c)

Figure 2: (a) A bipartite graph; (b) the table of crossings numbers; (c) the penalty digraph.

incident to v depends only on the relative position of u and v in the ordering x_0. Hence, for each $u, v \in L_0$ the *crossing number* c_{uv} can be univocally defined as the number of crossings between edges incident to u and edges incident to v when $x_0(u) < x_0(v)$. A trivial lower bound on the number of edge crossings for any fixed ordering x_1 can be defined as follows:

$$LB(G, x_1) = \sum_{u,v \in L_0} min\{c_{uv}, c_{vu}\} \tag{1}$$

It is easy to see that, given a permutation x_1, $LB(G, x_1)$ crossings are unavoidable in any bipartite drawing of G with layer L_1 ordered according to x_1: $\forall x_0 \; LB(G, x_1) \leq cross(G, x_0, x_1)$ and, in particular, $LB(G, x_1) \leq opt(G, x_1)$. Despite of the simplicity of Equation 1, it has been observed in [22] that the lower bound seems to be very tight, i.e., the value of $LB(G, x_1)$ is usually very close to $opt(G, x_1)$.

The concept of crossing number leads to the following definition of *penalty digraph*:

Definition 1 *[31] Let $G = (L_0, L_1, E)$ be a bipartite graph and let x_1 be an ordering of the vertices in L_1. The penalty digraph built from G and x_1 is a weighted directed graph $H = (N, A, w)$ such that: (a) $N = L_0$; (b) $A = \{(u, v) \in N \times N$ such that $c_{uv} < c_{vu}\}$; (c) $w : A \to \{1 \ldots \infty\}$ such that $\forall (u, v) \in A$, $w(u, v) = c_{vu} - c_{uv}$.*

A bipartite graph, the crossing numbers, and the corresponding penalty digraph are exemplified in Figure 2. The penalty digraph H has many interesting properties with respect to the crossing minimization problem. Indeed, the presence of an arc from u to v in H, due to the fact that $c_{uv} < c_{vu}$, means that we would like to have u before v in the ordering x_0. If this is not the case, exactly $c_{vu} - c_{uv} = w(u, v)$ more crossings are introduced. In particular, the following property can be easily proved:

Lemma 1 *Let $G = (L_0, L_1, E)$ be a bipartite graph and let x_1 be a permutation of the vertices in L_1. The penalty digraph $H = (N, A, w)$ built from G and x_1 is acyclic iff $opt(G, x_1) = LB(G, x_1)$.*

Proof: Let us first suppose that H is acyclic and let x_0 be any topological sort of its nodes. For any $u, v \in L_0$ such that $(u, v) \in A$, $x_0(u) < x_0(v)$; the contribution to the total number of crossings due to edges incident to u and to edges incident to v in G is then equal to c_{uv}. On the other side, for any $u, v \in L_0$ such that neither $(u, v) \in A$ nor $(v, u) \in A$, $c_{uv} = c_{vu}$ by construction of the penalty digraph and the relative ordering of u and v in x_0 is irrelevant. These considerations imply that $cross(G, x_0, x_1) = LB(G, x_1)$, forcing the optimum solution to have the same value as well.

Let us now assume that H contains a cycle. Under this hypothesis for any ordering x_0 of the vertices in L_0 there exist at least an arc $(u, v) \in A$ such that $x_0(u) > x_0(v)$. The contribution to the total number of crossings due to edges incident to u and to edges incident to v is then equal to $c_{vu} > c_{uv}$. These considerations apply in particular to the optimum solution, whose value is therefore strictly greater than $LB(G, x_1)$. $\square$

If the penalty digraph is acyclic, thanks to Lemma 1 any topological sort of the nodes of H is an optimal ordering x_0, i.e., an ordering x_0 such that $cross(G, x_0, x_1) = opt(G, x_1)$. This proves the polynomiality of CP on instances for which $opt(G, x_1) = LB(G, x_1)$. An immediate consequence concerns biplanar graphs, i.e., graphs which admit a planar bipartite drawing. Indeed, if $opt(G, x_1) = 0$, then a topological sort of the nodes of H yields a biplanar drawing:

Corollary 1 *If a bipartite graph G is biplanar for a given ordering x_1, then any topological sort of the nodes of the penalty digraph yields a biplanar drawing.*

Corollary 1 highlights for biplanarity (with a fixed layer) the same situation arising in general for planarity, i.e., it can be checked in polynomial time if a graphs is planar, but, in case it is not, its maximum planar subgraph cannot be efficiently found. In our case, determining if a graph has a biplanar subgraph with at least k edges when the ordering of a layer is fixed is NP-complete [16], yet if $k = m$ the problem is polynomially solved by the penalty minimization technique.

Unfortunately, in general the penalty digraph is not acyclic and does not admit any topological sort. In all these situations it is necessary to remove all directed cycles from H by cutting off some arcs. For each removed arc (u, v) the topological sort will then place v before u introducing $w(u, v) = c_{vu} - c_{uv}$ more crossings than if u would be placed before v. The sum of the weights of removed arcs is equal to the total number of exceeding crossings and should be kept as low as possible. In other words, we are interested in finding the *minimum feedback arc set* of the penalty digraph H. A feedback arc set of a digraph is a subset of its arcs containing at least one arc from every directed cycle. The minimum (weighted) feedback arc set problem can be stated as follows:

FAS: Given a directed graph $H = (N, A, w)$ with positive arc weights w, find a feedback arc set $F \subseteq A$ having minimum weight $W(F) = \sum_{(u,v)\in F} w(u, v)$.

The penalty minimization technique for solving CP proposed in [31] finds inspiration from the previous considerations: a bipartite drawing of G with the *minimum* number of crossings can be obtained by finding the *minimum* feedback arc set of the penalty digraph H and then by topologically sorting the nodes of H according to the ordering induced by the remaining arcs. The NP-completeness of the minimum feedback arc set problem [20] has strongly limited the diffusion of this technique: its practical utility has been for long time overlooked and the approach has been considered of theoretical interest only. However, some new results presented in this paper, both theoretical and experimental, show that the penalty minimization approach can be effectively used for minimizing crossings in bipartite drawings.

2.1 An approximation-preserving reduction from CP to FAS

Starting from the correspondence between the *minimum* feedback arc set of H and the minimum number of crossings of G claimed in [31], in this section we show a correspondence between *approximate* solutions of FAS and CP and we prove that the approximation ratio is preserved by the reduction. We first need to recall the definition of AP-reduction, referring the interested reader to [1] for basic definitions and concepts concerning approximability.

Definition 2 *[1] Given two optimization problems $\mathcal{P}_1$ and $\mathcal{P}_2$, we say that $\mathcal{P}_1$ is AP-reducible to $\mathcal{P}_2$ if two functions f and g and a constant $\alpha \geq 1$ exist such that:*

1. *for any instance x of $\mathcal{P}_1$ and for any $r > 1$, $f(x, r)$ is an instance of $\mathcal{P}_2$;*

2. *for any instance x of $\mathcal{P}_1$ and for any $r > 1$, if there exists a feasible solution of x with respect to $\mathcal{P}_1$, then there exists a feasible solution of $f(x, r)$ with respect to $\mathcal{P}_2$;*

3. *for any instance x of $\mathcal{P}_1$, for any $r > 1$, and for any solution y of $f(x, r)$ with respect to $\mathcal{P}_2$, $g(x, y, r)$ is a feasible solution of x with respect to $\mathcal{P}_1$;*

4. *f and g are computable in polynomial time for any fixed r;*

5. *for any instance x of $\mathcal{P}_1$, for any $r > 1$, and for any r-approximate solution y of $f(x, r)$ with respect to $\mathcal{P}_2$, $g(x, y, r)$ is a $1 + \alpha(r - 1)$-approximate solution of x with respect to $\mathcal{P}_1$.*

Theorem 1 CP *is AP-reducible to* FAS.

Proof: We have already shown that CP is reducible to FAS in polynomial time using the concept of penalty digraph. Now we prove that this reduction preserves the approximation according to the last point of Definition 2. In more detail, we show the existence of a constant $\alpha \geq 1$ such that any r-approximate solution of FAS on the penalty digraph, $r > 1$, can be transformed into a $1 + \alpha(r - 1)$-approximate solution of CP on the original graph. In particular, we show that $\alpha = 1$, that is the same approximation ratio is maintained by the reduction.

To this aim, let $G = (L_0, L_1, E)$ be a bipartite graph, let x_1 be the ordering of the vertices in L_1, and let $H = (L_0, A, w)$ be the penalty digraph built from G and x_1. We denote with F a generic feedback arc set of H and with F^* a minimum feedback arc set. Moreover, we denote with $c(F)$ the number of crossings in a bipartite drawing of G determined by a topological sort of the acyclic digraph $(L_0, A - F)$, i.e., the acyclic digraph obtained from H by removing the arcs in F.

Due to the correspondence between optimum solutions of FAS and CP claimed in [31], we have that $opt(G, x_1) = c(F^*) = LB(G, x_1) + W(F^*)$. In general, the following relation holds:

$$c(F) \leq LB(G, x_1) + W(F) \tag{2}$$

Indeed, any arc $(u, v) \in A - F$ always contributes with c_{uv} crossings, while any arc $(u, v) \in F$ may contribute with c_{vu} crossings. We remark that if $(u, v) \in A$ then $c_{uv} = min\{c_{uv}, c_{vu}\}$. The total exceeding contribution is therefore bounded by $W(F)$.

Let now r be any approximation ratio for FAS. Therefore, $W(F) \leq r \cdot W(F^*)$ holds. The following chain of inequalities:

$$c(F) \leq LB(G, x_1) + W(F) \leq LB(G, x_1) + r \cdot W(F^*) \leq r \cdot (LB(G, x_1) + W(F^*)) = r \cdot opt(G, x_1)$$

completes the proof and exhibits a constant α equal to 1. □

2.2 The relaxed penalty minimization scheme

Theorem 1 suggests that the penalty minimization approach can be used to devise approximation algorithms for the one-sided crossing problem by considering a *relaxed penalty minimization scheme* (in short, RPMS): given a bipartite graph $G = (L_0, L_1, E)$, after computing the crossing numbers of G, the penalty digraph $H = (N, A, w)$ is built; then, an approximation F of the minimum feedback arc set of H is computed and finally a topological sort x_0 of the nodes of the digraph $(N, A - F)$ is returned. The relaxation consists of accepting approximate solutions of FAS and this makes it possible to devise polynomial-time approximation algorithms for CP. An interesting property of RPMS algorithms is their capability of finding an optimum solution for instances for which $opt(G, x_1) = LB(G, x_1)$.

Complexity and approximation ratio obtained using RPMS depend on the complexity and on the approximation ratio of the algorithm used for finding a feedback arc set. So, any improvement or new result concerning FAS also implies an improvement to the corresponding algorithm for solving CP. In particular, the fact that $\alpha = 1$ in the proof of Theorem 1 means that the same approximation ratio as for FAS is obtained for CP. However, we want to point out that in the chain of inequalities which completes the proof we used a very trivial bound: $LB(G, x_1) \leq r \cdot LB(G, x_1)$. This fact, together with the experimental observation that $W(F)$ is usually smaller than $LB(G, x_1)$, allows us to assert that from an empirical point of view the approximation ratio for CP is much better than that for FAS.

The temporal complexity of the relaxed penalty minimization scheme is dominated by three main operations, i.e., computing the crossing numbers, building the penalty digraph H, and finding a feedback arc set of H. As we will detail in Section 3, crossing numbers can be computed in $O(m \cdot n_0)$ time. Moreover, we will show a method which allow us to find a feedback arc set working directly on the matrix containing the crossing numbers, thus avoiding the construction of H. In conclusion, if we denote with $f(H)$ the time spent for finding a feedback arc set of H, the overall complexity of RPMS is $O(max\{m \cdot n_0, f(H))\})$.

2.3 Removing cycles in the penalty digraph

This section focuses on the choice of the algorithm for computing a feedback arc set. Preliminary studies on the structure of the penalty digraph obtained from different instances of bipartite graphs led us to observe that it usually appears to be really dense and then to conjecture that it contains many short cycles. These observations have been later experimentally confirmed, as we report in Section 5. It is then natural to think of an algorithm able to take advantage of this special structure.

Among the different solutions known in literature [9, 14, 15, 18, 30], the algorithm described in [9] (in short, LR) seems a good choice, not just because it is faster than the algorithm achieving the best approximation ratio for FAS [18], but mostly because it obtains better results on directed graphs with short cycles. Namely, algorithm LR runs in time quadratic in the number of arcs of the digraph and approximates the best solution within a ratio bounded by the length (i.e., the number of arcs) of a maximum length simple cycle.

Algorithm LR is based on the local-ratio technique, that is a powerful yet simple tool for designing approximation algorithms [2] and works by progressively reducing the weights of the arcs of the digraph. In more detail, the algorithm looks for a cycle $\mathcal{C}$ in the digraph and, if it exists, identifies an arc in $\mathcal{C}$ having minimum weight, say ϵ. Then, the weight of all the arcs in $\mathcal{C}$ is decreased by ϵ and the arcs whose weight becomes equal to 0 are deleted. If the digraph is now acyclic LR terminates, otherwise the previous steps are repeated. The feedback arc set returned by LR is the set of deleted arcs. Since these steps do not guarantee the feedback arc set to be minimal, a second phase is performed trying to recover the minimality of the solution. In this phase the algorithm tries to add back some of the removed arcs, paying attention not to reintroduce cycles. The following theorem concerning algorithm LR can be proved:

Theorem 2 *[9] Let $H = (N, A, w)$ be a weighted directed graph and let m be equal to $|A|$. Algorithm LR finds a feedback arc set of H in time $O(m^2)$ and with a guaranteed approximation ratio bounded by the length λ of a longest simple cycle of H found by LR during its execution.*

The algorithm for CP that we empirically study in Section 5, called PM, hinges upon the relaxed

```
    procedure CrossTable (G = (L0, L1, E))
1.  begin
2.     for each u ∈ L0
3.        for each v ∈ L0 s.t. v > u
4.           c_uv, c_vu ← 0
5.           scan ← 0
6.           adj(u) ← first adjacent of u
7.           adj(v) ← first adjacent of v
8.           while adj(u) ≠ null
9.              while adj(v) ≠ null and adj(v) < adj(u)
10.                adj(v) ← next adjacent of v
11.                scan ← scan + 1
12.             c_uv ← c_uv + scan
13.             c_vu ← c_vu + degree[v] − scan − 1
14.             if adj(v) < adj(u) then c_vu ← c_vu + 1
15. end
```

Figure 3: Computing the cross table: the adjacency lists of vertices of L_0 have been previously sorted according to x_1

penalty minimization scheme and uses algorithm LR as a subroutine for computing the feedback arc set on the penalty digraph. Since the number of arcs of the penalty digraph is trivially bounded by n_0^2, according to Theorem 2 $f(H) = O(n_0^4)$. As stated in Theorem 1, algorithm PM has the same approximation ratio as algorithm LR: its performance ratio is therefore limited by the length of a longest simple cycle of the penalty digraph H, that is in turn bounded by n_0.

3 Implementing algorithm PM

Since a number of interesting engineering questions were encountered while implementing algorithm PM, in this section we highlight the most meaningful heuristics and implementation issues introduced for improving its performances.

First of all, we avoid explicitly building the penalty digraph. Indeed, according to Definition 1, an arc (u, v) exists in H if and only if $c_{vu} - c_{uv} > 0$. This implies that it is sufficient to hold the crossing numbers, that can be stored in a matrix called *cross table*.

Using a simple-minded approach, the cross table could be easily computed in $O(m^2)$ time. However, if the adjacency lists of the vertices of the bipartite graph are kept ordered according to x_1, it is possible to build it in $O(m \cdot n_0)$ time. Without loss of generality we can assume that any vertex of L_1 considered during the computation of the cross table has degree at least one: such vertices can be identified in a preprocessing step. This implies that $m \geq n_1$ and that $O(m)$ time is sufficient to sort the adjacency list of a vertex of L_0 using a linear sorting algorithm (e.g., counting sort [6]). Sorting all the adjacency lists requires then $O(m \cdot n_0)$ time in total. An additional $O(m \cdot n_0)$ time is sufficient to compute the crossing numbers: actually, the adjacency list of each vertex can be scanned only n_0 times thanks to the pre-computed ordering. The pseudo-code related to this phase is given in Figure 3. It is easy to see that the running time of the loops in line 8 and line 9 is $O(degree_u + degree_v)$. Therefore, the total complexity is given by

$$\sum_{u \in L_0} \sum_{v \in L_0} (degree_u + degree_v) = \sum_{u \in L_0} (n_0 \cdot degree_u + m) = O(m \cdot n_0)$$

As we do not explicitly store the penalty digraph in memory, we require a practical way for keeping track of the decrements of the weights of its arcs. We choose to progressively modify the content of the cross table, paying attention not to destroy relevant "semantic" properties of the table itself. In particular, if $\mathcal{C}$ is the cycle found by the algorithm at a generic iteration and (s, e) is a minimum weight arc in $\mathcal{C}$, for each arc $(x, y) \in \mathcal{C}$ the cross table entries c_{xy} and c_{yx} are increased by c_{es} and c_{se}, respectively. By means of simple calculations, it is possible to verify that these assignments guarantee arcs in $\mathcal{C}$ to preserve their previous direction and their weights to be correctly decreased by $w(s, e)$.

In more detail, let us call $\tilde{c}_{xy}$ and $\tilde{c}_{yx}$ the values $c_{xy} + c_{es}$ and $c_{yx} + c_{se}$, respectively, i.e., the values of the cross table entries after the update. Let also $\tilde{w}(x, y)$ be the weight of arc (x, y) after the update. We have that

$$\tilde{w}(x, y) = \tilde{c}_{yx} - \tilde{c}_{xy} = c_{yx} + c_{se} - c_{xy} - c_{es} = w(x, y) - w(s, e) \geq 0$$

because (s, e) is a minimum weight arc in $\mathcal{C}$. Minimum weight arcs of $\mathcal{C}$ correctly disappear from the penalty digraph as their new weight $\tilde{w}(x, y)$ becomes equal to 0.

In order to speed up the execution of algorithm LR, we also implement it in order to look for more than one cycle during the same visit. In more detail, each depth-first visit of the digraph does not stop when a cycle is found: on the contrary, the cycle is broken (this requires an operation linear in the length of the cycle itself) and the visit goes on to find another cycle. Although the worst case temporal complexity appears not to change, this choice allows us to consistently reduce the number of visits and results in a substantial improvement of the running time of algorithm LR from an empirical point of view.

4 Experimental setup

In this section we describe our experimental framework, presenting the algorithms being compared, the problem instances generators, the performance indicators we consider, as well as some implementation details. All implementations have been realized by the authors in ANSI C and have been debugged and tested by using the logic-based software visualization system Leonardo [7], available over the Internet at the URL `http://www.dis.uniroma1.it/~demetres/Leonardo/`.

Experimental tests have been performed on a Macintosh PowerBook G3 at 293 Mhz with 64 MB of main memory running Linux Kernel 2.2.6 and all programs have been compiled through the GNU gcc compiler version 1.1.2 with optimization level O4. The full package, including algorithm implementations, graph generators, visualization code, and a test program, is available by the authors over the Internet at the URL: `http://www.dsi.uniroma1.it/~finocchi/experim/cplib-2.0/`.

4.1 Algorithms under evaluation

We chose the algorithms to analyze experimentally both for their novelty and according to previous computational results. In addition to algorithm PM, we address three well-known methods, i.e., the two "averaging" algorithms (BAR and MED) and the stochastic heuristic (STOC), and two recent and attractive strategies (GRE and SIFT). Indeed, Jünger and Mutzel show that, among the algorithms they study in [22], BAR, STOC and the SPLIT heuristic described in [13] are better at minimizing crossings than the other ones; however, since SPLIT in some cases takes even more time than computing the optimum solution, we did not implement it. Another promising approach that we contemplate in our package is mentioned in [11] and is in line with the ideas suggested by Gansner *et al.* [19]. In the sequel we review in more detail the main features of each algorithm.

The *barycenter* heuristic (BAR) assigns the x-coordinate of each vertex in L_0 as the average of the x-coordinates of all its neighbors [4, 31]. It runs in linear time and its approximation ratio is equal to $\sqrt{n_1}$ [17]. Similarly, the basic *median* strategy (BMED) assigns the x-coordinate of each vertex in L_0 as the x-coordinate of the median neighbor of that vertex [17]. It approximates the optimum solution within a constant ratio equal to 3, but it appears not to return accurate solutions in practice [22]. For this reason, in our experimentation we consider an enhanced version (MED or EMED) of the median algorithm, that does not run in linear time but produces better solutions [11, 19]. MED assigns the x-coordinates according to the median strategy but, if two vertices overlap, tries to split them up using the barycenter positions. At last, a sorting strategy is used for further improving the quality of the solution: vertices in L_0 are repeatedly scanned from left to right and two consecutive vertices u and v are swapped if $c_{uv} > c_{vu}$. MED runs in quadratic time due to the final bubblesort-like improvement phase.

A heuristic method for crossing reduction based on a stochastic approach (STOC) is devised in [12]. First, a $n_0 \times n_1$ frequency matrix is computed: each item f_{ij} of this matrix represents the number of edges that cross an edge drawn between position i and position j in a complete bipartite graph. A $n_0 \times n_0$ table of assessment numbers is then obtained from the frequency matrix: each item a_{ui} of the table measures the effect of placing vertex $u \in L_0$ in position i and is computed as the geometrical mean of those elements of the frequency matrix corresponding to an edge in the adjacency vector of vertex u. Vertices are finally placed in a greedy like fashion based on the assessment numbers. Our implementation of the stochastic heuristic runs in $O(n_0^2 \cdot n)$ time due to the assessment numbers computation and to the repeated minimum extraction.

Recently, a greedy approximation algorithm (GRE) has been proposed [34]. This algorithm incrementally builds up an ordering for L_0 by placing at each step the vertex that minimizes a simple heuristic function; such function depends on the number of crossings introduced by the vertex with respect to not yet placed vertices. The greedy algorithm achieves an approximation ratio that varies from 2 to 3, according to the maximum degree of the graph, and theoretically outperforms all the other algorithms. The good quality of the solutions found by GRE is paid in terms of its temporal complexity, i.e. $O(max\{m^2, n_0^2\})$.

At last, a heuristic for k-layer straightline crossing minimization which can be easily adapted to CP is described in [25]; the heuristic, called sifting (SIFT), was first used for dynamic reordering of decision diagrams during synthesis and verification of logic circuits [29]. At each step, a vertex v to be sifted is chosen from L_0. Sifting on v consists of identifying a locally optimal position for v, while keeping the position of any other vertex fixed: this reduces to repeatedly swapping v with its left/right neighbor. The next vertex to be sifted is chosen according to a precomputed ordering: as suggested by the authors, in our code we sift vertices with high degree first. SIFT can be implemented to run in $O(m \cdot n_0 + n^2)$ time, using crossings numbers to recompute the number of crossings after each swap.

Notice that not all the algorithms are based on crossing numbers: BAR and STOC are completely independent from this concept, MED uses crossing numbers only in its second phase in order to refine the solution, while both GRE and SIFT (in addition to PM) strongly depends on them. Computing crossing numbers slows down the computation, though algorithms based on their use always produce better solutions than the other ones, as we show in Section 5. We remark that the same code for computing crossing numbers is used in the package by all the algorithms which require them.

To test the effectiveness of the aforementioned algorithms in a more general context, we also implemented a simple parameterized version of the hierarchical method [31] which can use any of them as a subroutine. After removing cycles with a depth-first visit, we layer vertices by means of a topological sort and we possibly add dummy vertices to ensure edges to span only consecutive layers. Starting from an arbitrary permutation of the vertices on the first layer, we follow a single-pass layer-by-layer sweep method where we solve a one-sided crossing problem on each pair of consecutive layers. We use as fixed permutation at each step the ordering returned by the CP subroutine at the previous step.

4.2 Performance indicators

Important objectives that we considered to evaluate the performance of algorithms include the quality of the obtained solution and the time spent for computing it.

We measure the quality of the solution not directly by means of the number of crossings, but with respect to the value of the lower bound $LB(G, x_1)$. Indeed, as CP is NP-complete, it is not possible to compute exact optima except for small graph instances [22]. However, since $LB(G, x_1)$ crossings are unavoidable in any drawing, we can compute an upper bound on the number of errors of an algorithm by subtracting the value $LB(G, x_1)$ from the number of crossings obtained by that algorithm. We refer to this measure as the *error upper bound*.

The running times of codes that we compare are reported in seconds (unless otherwise stated) and have been computed using the UNIX system call `getrusage()`. Since on many small and sparse instances the granularity of time measuring on our platform (60 ticks per second) did not allow us to compute confident measures, we cycled the run of each algorithm on the same data up to a total time greater than 0.1 seconds, and we averaged the running time on the number of runs.

In all the experiments we also took into account some structural measures on the problem instances that we consider to be relevant for a better comprehension of the problem and of the behavior of algorithm PM. These measures are related both to the input bipartite graph and to combinatorial properties of the penalty digraph:

- *LB*: this measure represents the trivial lower bound on the number of crossings and is computed according to Equation 1.

- $\frac{UB}{LB}$: as in Equation 1, an upper bound on the number of crossings can be obtained as follows:

  $$UB(G, x_1) = \sum_{u,v \in L_0} max\{c_{uv}, c_{vu}\}$$

  $\frac{UB}{LB}$ is a measure that reveals the intrinsic difficulty of a problem instance: it can be thought as the approximation ratio guaranteed by the worst algorithm for CP on that instance. To cope with the case where $LB = 0$, we actually report the quantity $\frac{UB+1}{LB+1}$. Due to theoretical considerations [11], we expected the sparser the instances, the higher this measure.

- *Penalty density*: we study the density of the penalty digraph as a function of the parameters characterizing the bipartite graph. This is interesting for understanding the structure of the directed graphs that the subroutine for FAS has to deal with.

- *Broken cycles*: we count the number of cycles found and removed by algorithm LR during its execution on the penalty digraph. The more cycles are broken, the more crossings are introduced with respect to the lower bound and the more time is required to compute a feedback arc set if compared to the other steps performed to solve CP. Although this measure strictly depends upon the LR subroutine, it provides interesting statistical clues about the closeness of the optimum to LB.

- *Max cycle length*: we measure the maximum length of a cycle broken by algorithm LR during its execution on the penalty digraph. This quantity is an upper bound on the approximation ratio guaranteed by algorithm PM on the input bipartite graph.

- *Average cycle length*: we sum up the length of the cycles broken by algorithm LR during its execution on the penalty digraph. We then average this number on the number of broken cycles. Due to preliminary considerations on the density of the penalty digraph, we conjectured this quantity to be quite small.

4.3 Synthetic test sets

Synthetic test sets are extremely important in the empirical assessment of algorithms as they make it possible to study performances with respect to variable input parameters such as problem size or edge density. Ad-hoc families of problems can also be targeted to reveal or confute desired properties of algorithms or to clearly separate their behavior.

Random graph generators. Random problem instances used to compare the algorithms have been produced by means of two bipartite graph generators.

- *Edge density* (ED): this random generator takes three input parameters n_0, n_1, and a number $\rho \in [0, 1]$ that represents the probability of generating an edge between any two vertices. For each possible edge, ED generates a random number $x \in [0, 1]$ and adds the edge to the graph if $x < \rho$. The expected number of edges is $\rho \cdot n_0 \cdot n_1$.

- *Maximum degree* (MD): this random generator takes three input parameters n_0, n_1, and an integer d representing the maximum degree of a vertex. For each vertex $u \in L_0$, MD generates a random number $d_u \in [0, min\{\tilde{n}_1, d\}]$, where $\tilde{n}_1$ is the number of vertices of L_1 currently having degree strictly smaller than d. Among them, d_u neighbors for u are then randomly chosen and $\tilde{n}_1$ is updated. Note that MD limits the degree of vertices both in L_0 and in L_1.

We remark that we used only odd seeds to initialize the random generators. Moreover, the sequence of seeds used in each test was itself randomly generated starting from a base seed. We used as subroutine the `rand()` pseudo-random source of numbers provided by the ANSI C standard library.

Warfield instances. Warfield instances [33] have already been considered in previous experimental studies [12, 22] and are regarded as particularly meaningful for CP since L_0 has exponential size with respect to L_1. We implemented a generator (WA) for Warfield instances which takes as input the size $k = n_1$ of the fixed layer and creates a bipartite graph such that $n_0 = 2^k - 1$ and the adjacency matrix is a $n_1 \times n_0$ matrix such that column j contains the binary representation of number j.

Solution structure. Interesting problem classes in experimental algorithmics include instances having a solution structure known in advance. This is particularly useful in the case of hard problems, where exact optima are difficult, or even impossible, to find efficiently. A common technique for generating instances with a given solution structure consists of starting from the desired solution and adding data to hide it [21].

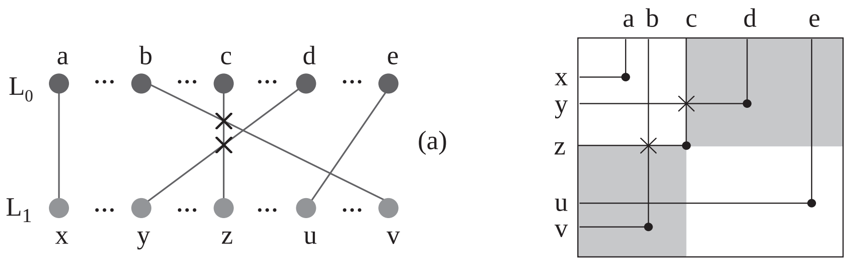


Figure 4: (a) Crossings in a bipartite drawing; (b) "crossings" in a structure plot.

In the case of CP, we generated bipartite graphs G and permutations x_1 of the fixed layer with the property that $opt(G, x_1) = LB(G, x_1)$, for which CP becomes polynomial as follows from Lemma 1. Our methodology was based on considerations relating crossing minimization to band diagonalization of matrices in finite element problems [4]. Figure 4 shows the correspondence between crossings in a bipartite drawing and "crossings" in a structure plot of a bipartite graph G induced by permutations x_1 and x_0: any dots in the grey regions depicted in Figure 4(b), such as the dots related to edges (v, b) and (y, d), cause a crossing with edge (z, c) in a bipartite drawing. In order to minimize crossings, we would like to find a permutation x_0 of the columns of the structure plot which minimizes the number of dots falling in the grey regions of any dot. This observation suggested us to produce structure plots exhibiting strong diagonal structure, and then to extract bipartite graphs from them considering row permutations as the x_1's. A typical synthetic example for which $opt(G, x_1) = LB(G, x_1)$ is shown in Figure 5. In order to hide the optima, we randomly permuted the columns of the structure plots before extracting the bipartite graphs.

Structure plots of some of the test sets we considered are shown in Figure 6. The sparse problem instance in Figure 6(a) is a degenerate case for which $opt(G, x_1) = LB(G, x_1) = UB(G, x_1)$, i.e., any algorithm yields solutions with the same number of crossings. Other typical examples of bipartite graphs for which $opt(G, x_1) = LB(G, x_1)$ are Warfield instances for $k = 3, 4, 5$.

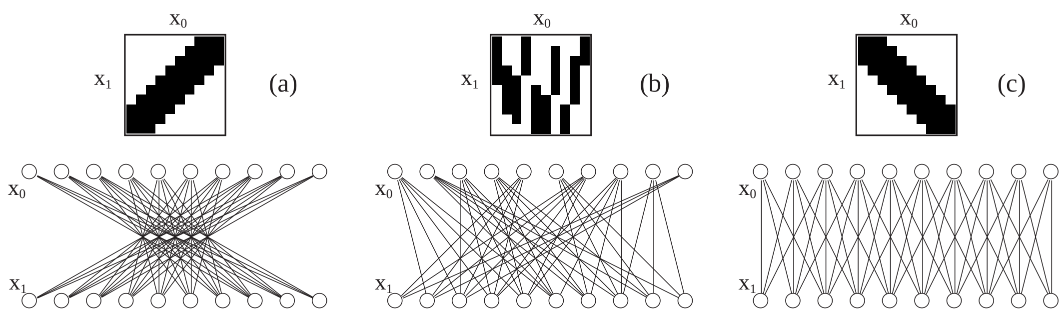


Figure 5: Structure plots and bipartite drawings of a 2-layered graph G: (a) x_0 s.t. $Cross(G, x_0, x_1) = UB(G, x_1)$; (b) random x_0; (c) x_0 s.t. $Cross(G, x_0, x_1) = LB(G, x_1)$.

4.4 Real test sets

In addition to synthetic instances, we also considered graphs obtained from real applications. Namely, we collected bipartite graphs both extracted from digitized images regarded as bipartite adjacency matrices as proposed by Knuth in the Standford Graph Base [23], and from test data retrieved from the US National Institute of Standards and Technology [3].

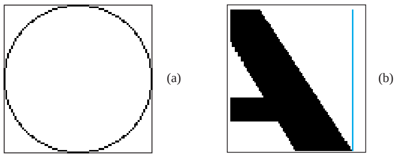


Figure 6: Test sets with desired solution structure $opt(G, x_1) = LB(G, x_1)$.

Digitized images. Digitized versions of rectangular paintings and images can be used to create bipartite graphs as suggested in [23]. As for structure plots, if h is the width and k is the height of the image, the sides L_0 and L_1 of the bipartition will contain h and k vertices, respectively. An edge between two vertices u and v exists iff the pixel in position $[u, v]$ in the $h \times k$ image has a value smaller than a given threshold. In other words, the binarized image can be interpreted as the adjacency matrix of a bipartite graph.

In addition to some portraits and to some textures exhibiting more or less regular patterns, in our experiments we also considered the famous Leonardo da Vinci's painting, Mona Lisa, which has been often used to explore combinatorial algorithms.

NIST sparse matrices. Matrix Market is a visual repository of test data for use in comparative studies of algorithms, with special emphasis on numerical linear algebra. It is a service provided by the US National Institute of Standards and Technology (NIST) and features nearly 500 sparse matrices from a variety of application domains, such as air traffic control, astrophysics, biochemistry, circuit physics, computer system simulation, finite elements, demography, economics, fluid flow, nuclear reactor design, oceanography, and petroleum engineering. Each matrix in the collection features a web page which provides statistics on the matrix properties and different visualizations of its structure (e.g., 2-dimensional structure plots or 3-dimensional city plots). We accessed and converted several sparse matrices from the collection into bipartite graphs regarding them as bipartite adjacency matrices where non-zero values are interpreted as edges. Matrices to use in our experiments were chosen according to a variety criteria, considering different sizes, densities, and symmetry parameters. Some examined matrices, such as the one related to US economic transactions in 1972 between commodities and industries (BEAUSE, see Figure 22), intrinsically represent bipartite graphs from the real world.

Other test suites. We considered a collection of directed graphs submitted to the e-mail graph drawing service at Bell Labs and frequently used as testbeds in the graph drawing field [27]. This repository consists of a single file containing more than 5000 general directed graphs. Moreover, we used graph instances which come from the Leonardo's repository of animated algorithms [8].

5 Experimental results

5.1 Randomly generated test sets

We performed four experiments on randomly generated graphs using different graph generators and parameters values. In all the experiments we averaged each data point on a number of trials varying with the density of the graph: we used 100 trials for the sparsest graphs ($\rho = 0.001$) and we decreased this number down to 10 trials for the most dense instances ($\rho = 0.5$) which exhibit lower variance in the measures.

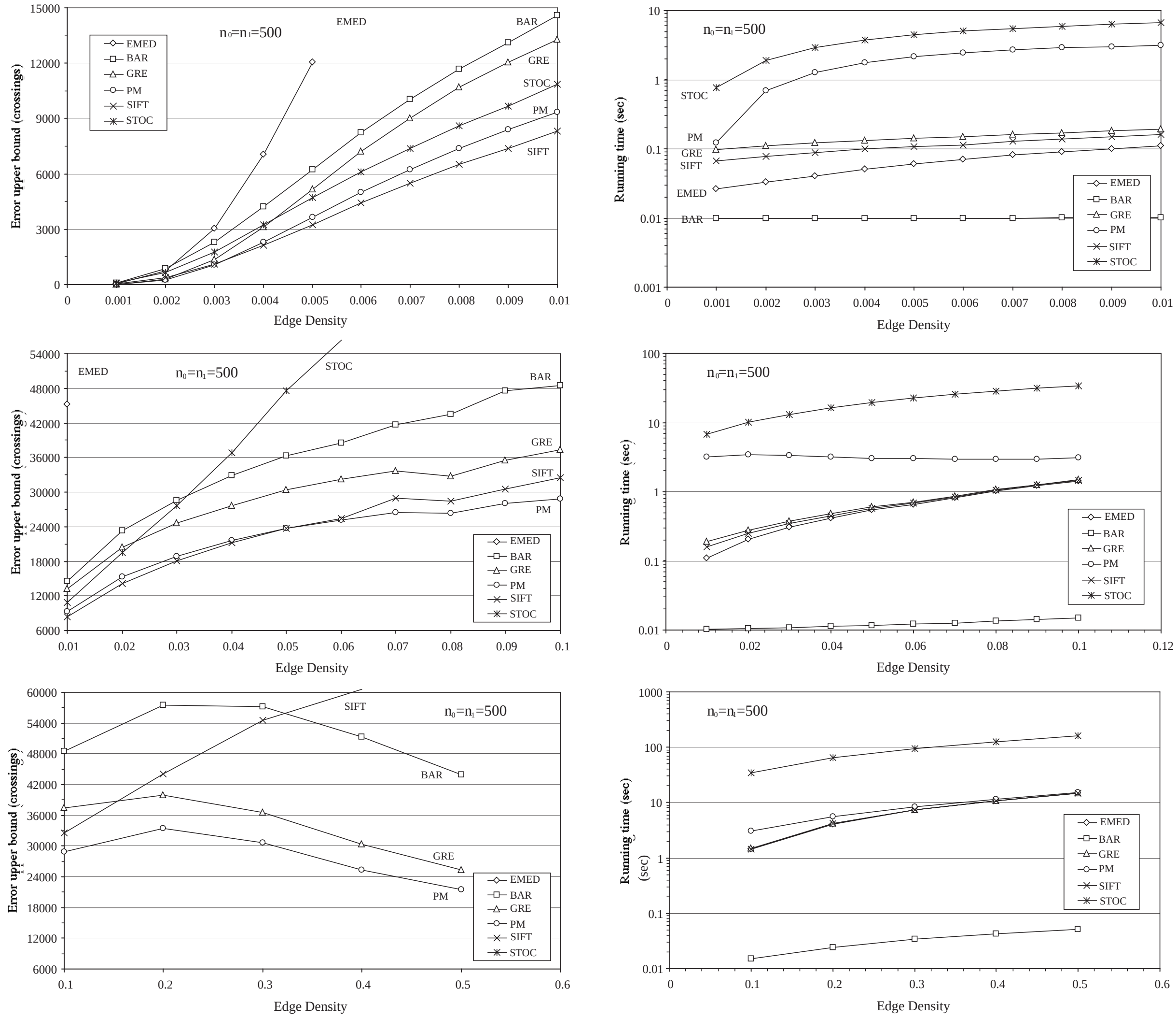


Figure 7: Error upper bound and running time for increasing edge density: $n_0 = n_1 = 500$ and $0.001 \leq \rho \leq 0.5$. The charts on each row show the results for different edge density intervals.

5.1.1 Increasing edge density

The aim of this experiment is to study how the performances of the algorithms depend on the density ρ of the bipartite graphs. Graph instances are generated using ED; the total number of vertices is fixed to 1000, the layers have the same size, and ρ increases from 0.001 to 0.5. We consider in more detail sparse instances because they are both the most frequent in real applications and the most difficult and meaningful with respect to CP: Figure 7 reports the error upper bound and running time for the three different ranges of density $[0.001, 0.01]$, $[0.01, 0.1]$, and $[0.1, 0.5]$.

With respect to the quality of the solutions, SIFT and PM appear to be the best algorithms: in particular, SIFT outstands in the range 0.004 to 0.05, while PM is better elsewhere. It is remarkable that, though CP is easier on dense instances due to combinatorial properties [11], not all the algorithms are able to exploit this property: PM, BAR, GRE and MED exhibit a decreasing trend for $\rho \geq 0.2$, while the error upper bounds for SIFT and STOC keep on increasing.

With respect to the running times, the higher is the number of edges, the higher is the time

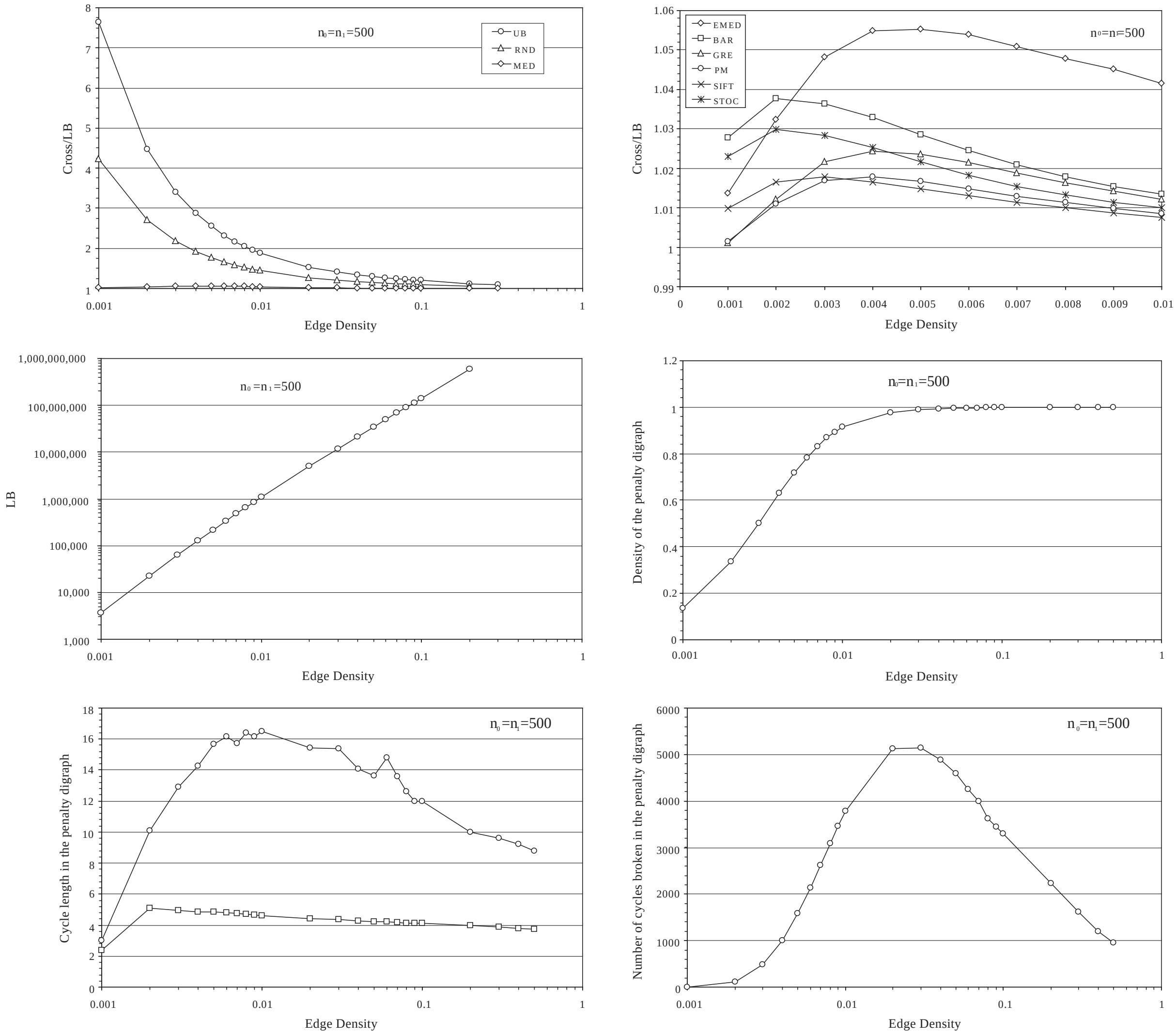


Figure 8: Structural measures for increasing edge density: $n_0 = n_1 = 500$ and $0.001 \leq \rho \leq 0.5$.

spent by all the algorithms. A surprising exception concerns the decreasing behavior of PM limited to the range 0.02 to 0.08. After noticing this fact, we decided to separately analyze the time required for computing the cross table and for finding a feedback arc set: we observed that the computation of crossing numbers becomes progressively more expensive as the density increases, according to the theoretical analysis, while the execution time of algorithm LR increases in $[0.001, 0.02]$ and decreases in $[0.02, 0.5]$. Since the total time decreases only in the range $[0.02, 0.08]$, it follows that the most expensive operation for algorithm PM up to density 0.08 is the computation of a feedback arc set, while computing the cross table becomes predominant on denser instances. Observe that all the algorithms maintain the same relative temporal performance ordering throughout the whole density range and that STOC is by far the slowest algorithm.

The analysis of the running time of algorithm PM is also supported by the measures on the penalty digraph, as shown in Figure 8. In particular, the density of the penalty digraph very rapidly grows in the range 0.001 to 0.01, up to reach its maximum value, i.e., 1. On the other side, the curve related to number of cycles broken by algorithm LR exhibits a non-monotonic aspect, with maximum value exactly in 0.02, thus explaining the observed improvement in the running time of PM. In addition to this, we remark the small length of the cycles removed by LR to make the penalty digraph acyclic: the values of both the average and the maximum length vary in a very small range, proving the feasibility of algorithm LR as subroutine for RPMS, and become smaller as ρ increases, confirming again the easiness of dense instances.

Figure 8 also shows an exponential growth of the value of the lower bound on random instances as the edge density increases: this greatly limits the chances of producing readable bipartite layouts of dense graphs. Fortunately, graphs from most real applications are very sparse, making the problem worthy of study. Nevertheless, the visual comparison of the drawing produced by any algorithm with the drawing obtained by a random choice of permutation x_0 clearly proves that minimizing edge crossings is very useful even on dense instances, as clearly shown in Figure 21. This is also evident from a numerical point of view: the curves related to the ratio between the number of crossings and the lower bound, illustrated in Figure 8, suggest that a random permutation obtain roughly half crossings than the maximum possible number, given by UB. Any crossing minimization algorithm obtains solutions really close to the lower bound. This fact confirms on a broader scale a consideration already reported in [22] for small instances: the lower bound and the optimum are extremely close. Indeed, though we cannot compute exact optima to compare the lower bound with, we can assert it by observing that $\frac{Cross}{LB}$ is very close to 1 for any algorithm and that the value of the optimum solution cannot be greater than the number of crossings obtained by the best algorithm.

5.1.2 Increasing number of vertices

The aim of this experiment is to study the behavior of the algorithms when the number of vertices increases and the edge density is kept fixed. The experiment was performed on bipartite graphs with equally sized layers randomly generated using ED with different edge densities. The total number of vertices $n = n_0 + n_1$ varies from 100 up to 1000. Figure 9, Figure 10, and Figure 11 show the results obtained for ρ equal to 0.001, 0.01, and 0.1, respectively.

In all the charts the error upper bound increases with the size of instance, though the relative performances of the algorithms are not same, according to the results obtained for increasing edge density. The error upper bound chart in Figure 9 details that PM and GRE behave better than SIFT on very sparse instances; this was not deducible from the corresponding chart in Figure 7. MED appears to be negatively affected by the size of the instance: this is quite reasonable, because for big instances the bubblesort-like improvement phase is more likely to incur in local optima. It is surprising that for $\rho = 0.01$ STOC, which usually does not return accurate solutions, minimizes crossings even better than the greedy algorithm.

The running times charts for the different sizes and densities highlight a quite tedious situation: BAR and STOC are the fastest and the slowest algorithms, respectively, while MED, SIFT, and GRE require very similar times. More interesting is the running time of algorithm PM, especially on small sparse instances: actually, when $\rho = 0.001$, PM is faster than GRE on instances with up to 400 vertices per layer. As the density increases, the intersection between the running time curves of PM and GRE shifts towards smaller instances (e.g., slightly more than 100 vertices for $\rho = 0.01$), and finally disappears when the edge density is 0.1. On bigger instances, the difference between the running times of our implementations of PM and GRE decreases as the edge density grows. This strengthens the theoretical analysis: from the asymptotical standpoint, PM and GRE have the same temporal complexity on dense graphs, but the more sparse is the graph, the more efficient is GRE with respect to PM.

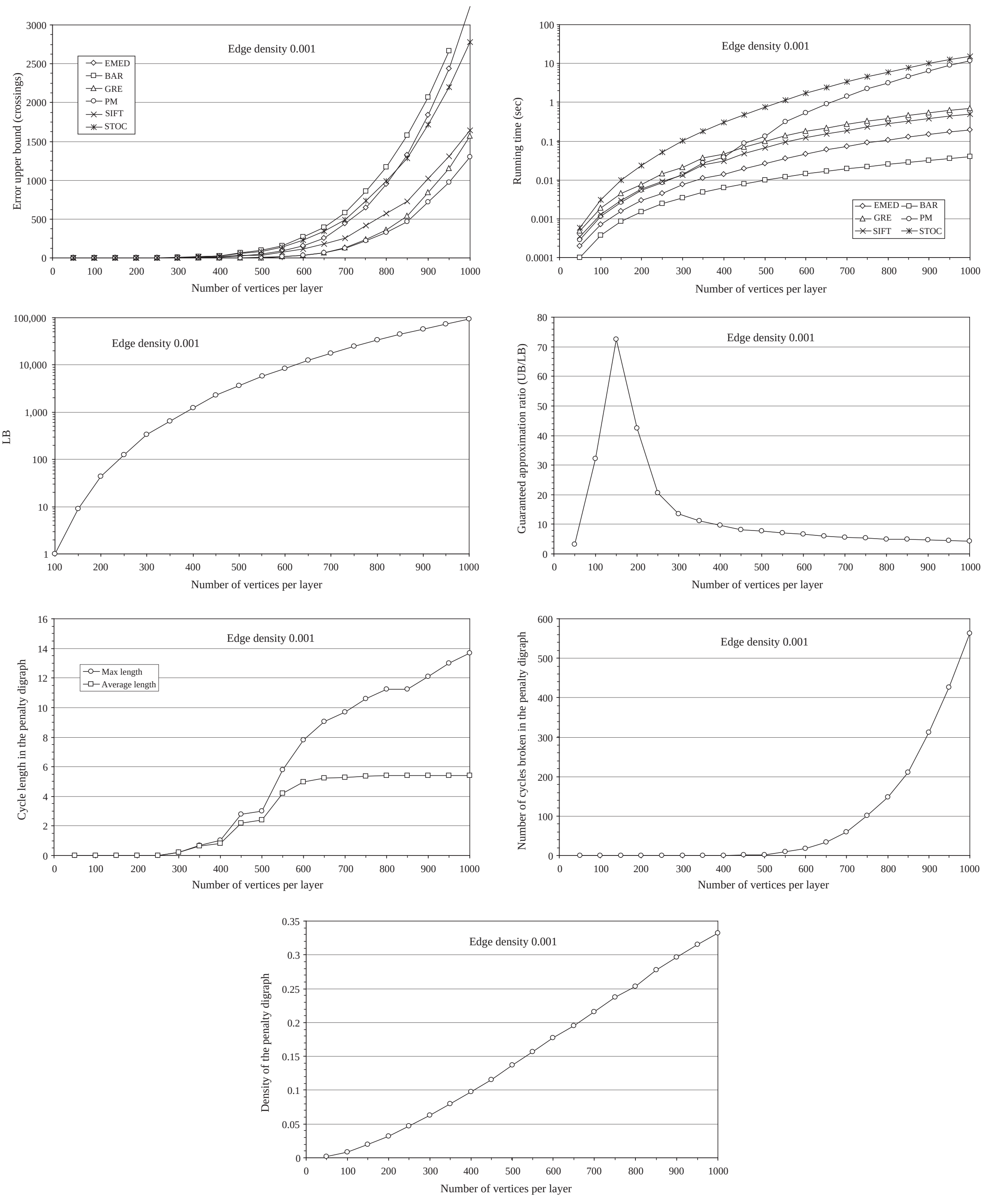


Figure 9: Increasing number of vertices: $50 \leq n_0 = n_1 \leq 1000$ and $\rho = 0.001$.

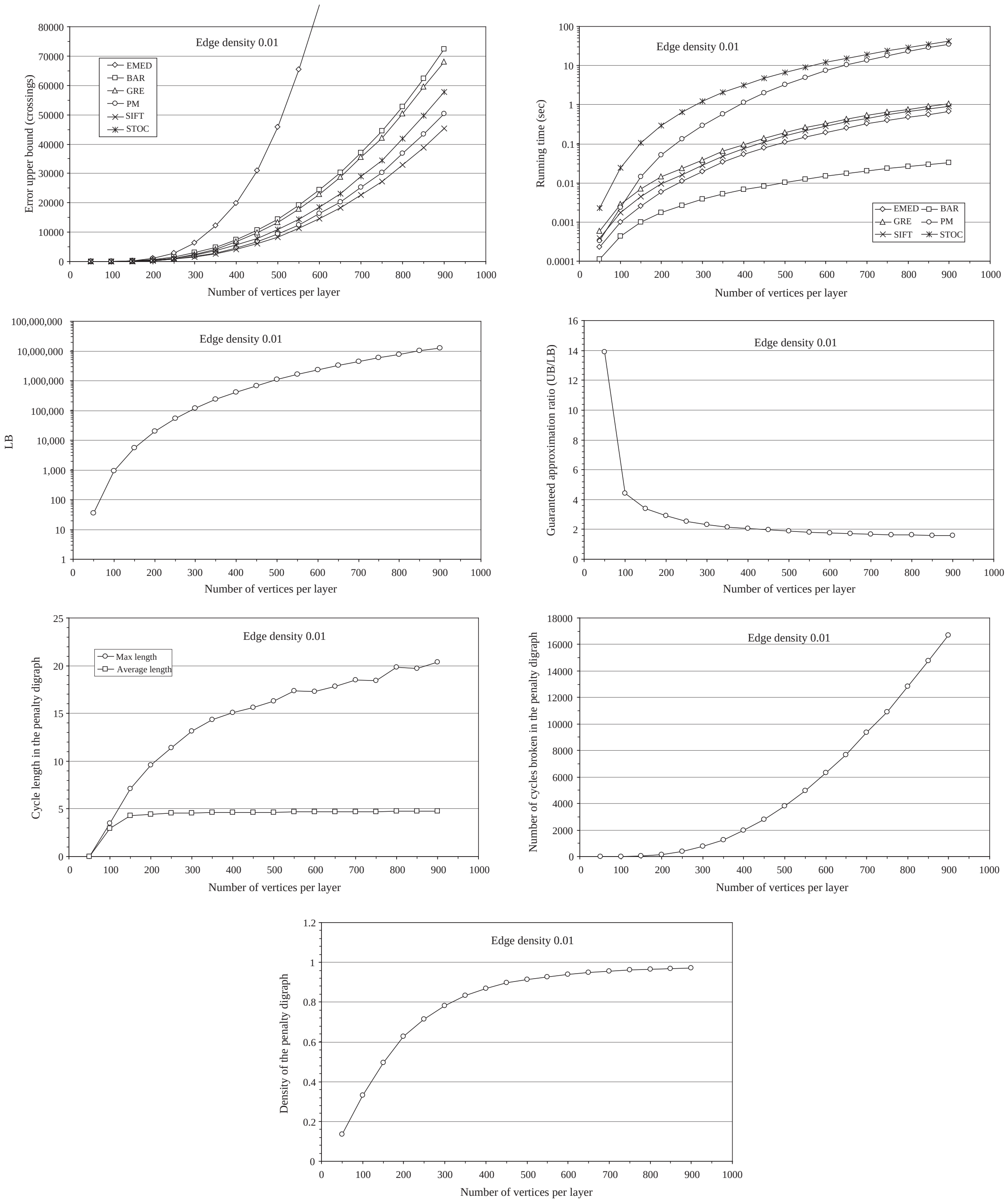


Figure 10: Increasing number of vertices: $50 \leq n_0 = n_1 \leq 900$ and $\rho = 0.01$.

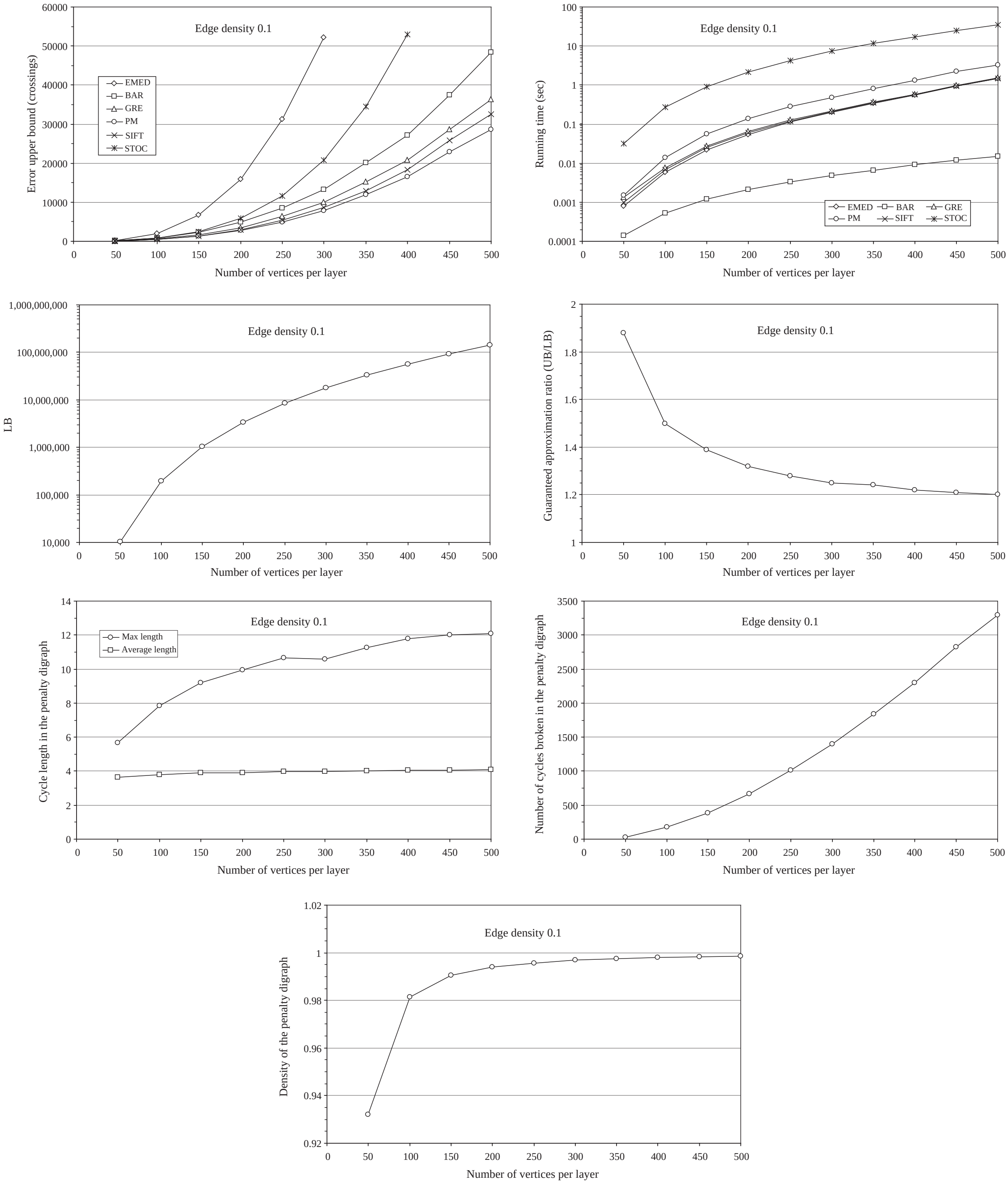


Figure 11: Increasing number of vertices: $50 \leq n_0 = n_1 \leq 500$ and $\rho = 0.1$.

Similarly to the previous experiment, we found that the value of the lower bound increases as the number of vertices does; however, in this case the growth rate appears to be subexponential, and therefore it is not as fast as for increasing edge density: this sounds well-grounded, because by definition the lower bound is much more affected by the number of edges than by the number of vertices in the bipartite graph. The trend of the curves related to $\frac{UB}{LB}$ depends on the fact that on small and sparse instances the value of the lower bound can very small: we empirically observed that in several cases it is exactly 0. Obviously, this is more unlikely to happen when the number of edges is bigger, as proved by the fact that the maximum value for $\frac{UB}{LB}$ is approximately 73 if $\rho = 0.001$, 14 if $\rho = 0.01$, and drops down to 1.9 if $\rho = 0.1$.

For what concerns the penalty digraph, it is interesting to observe that its density increases with the number of vertices in the bipartite graph, even if the edge density ρ is kept fixed. At a first glance this could appear strange, but can be explained by observing that increasing n_1 implies increasing the average degree of vertices of L_0, which is equal to $\rho \cdot n_1$. As a consequence, crossing numbers take their values in a larger range, and it is more likely for any pair c_{uv} and c_{vu} to differ by one unit: more arcs are therefore created in the penalty digraph. The number of cycles broken by algorithm LR consequently increases, yet their length remains small; in particular, note that the average cycle length exhibits an almost constant tendency as the instance size increases.

5.1.3 Increasing layers size ratio

This experiment aims at studying how the algorithms are affected by the relative size of the two layers of the input graphs. The total number of vertices is fixed to 1000, and the ratio $\frac{n_0}{n_0+n_1}$ increases from 0.02 to 0.98. Instances for this experiment are generated using ED for the three edge densities $\rho = 0.001$, $\rho = 0.01$, and $\rho = 0.1$. The results for each density are shown in Figure 12, Figure 13, and Figure 14, respectively.

Notice that, if the edge density ρ is kept fixed, the expected number of edges of each randomly generated bipartite graph is $m = \rho \cdot r(1-r)(n_0+n_1)^2$ where $r = \frac{n_0}{n_0+n_1}$. Such function is maximum when $r = \frac{1}{2}$, that is $n_0 = n_1$. This explains the non-monotonic aspect of the curves, and in particular the trend of LB.

Comparing the three charts related to the number of cycles broken by PM in the penalty digraph, it is interesting to observe that the maximum is reached for values of r which increase with the edge density ($r \approx 0.4$ for $\rho = 0.001$, $r \approx 0.65$ for $\rho = 0.01$, and $r \approx 0.85$ for $\rho = 0.1$). This behavior is confirmed not only by the error upper bound of PM, but also by the error upper bound of the other algorithms and by the average and maximum cycle length. For very sparse random instances of density about 0.001, the value of r for which the problem is most difficult is obtained for $n_0 \leq n_1$. For more dense instances, the same holds for $n_0 \geq n_1$.

As a further confirmation of the importance of considering the edge density for comparing the relative performance of the algorithms, the three charts for ρ equal to 0.001, 0.01, and 0.1 reveal completely different scenarios for the error upper bound. See, for instance, the behavior of BAR and MED. Again, the same cannot be said about the relative running time performances.

The fact that the more different is n_0 from n_1, the easier is the problem, and the growth of UB/LB for increasing r suggest that the advantage of using a crossing minimization algorithm instead of a random permutation x_0 increases with r.

Also note that the density of the penalty digraph decreases as r grows. This is explained by the fact that the average degree of any pair of vertices $u, v \in L_0$ decreases as L_1 shrinks, thus forcing crossing numbers c_{uv} and c_{vu} to get smaller. For this reason, the probability of having $c_{uv} = c_{vu}$ increases.

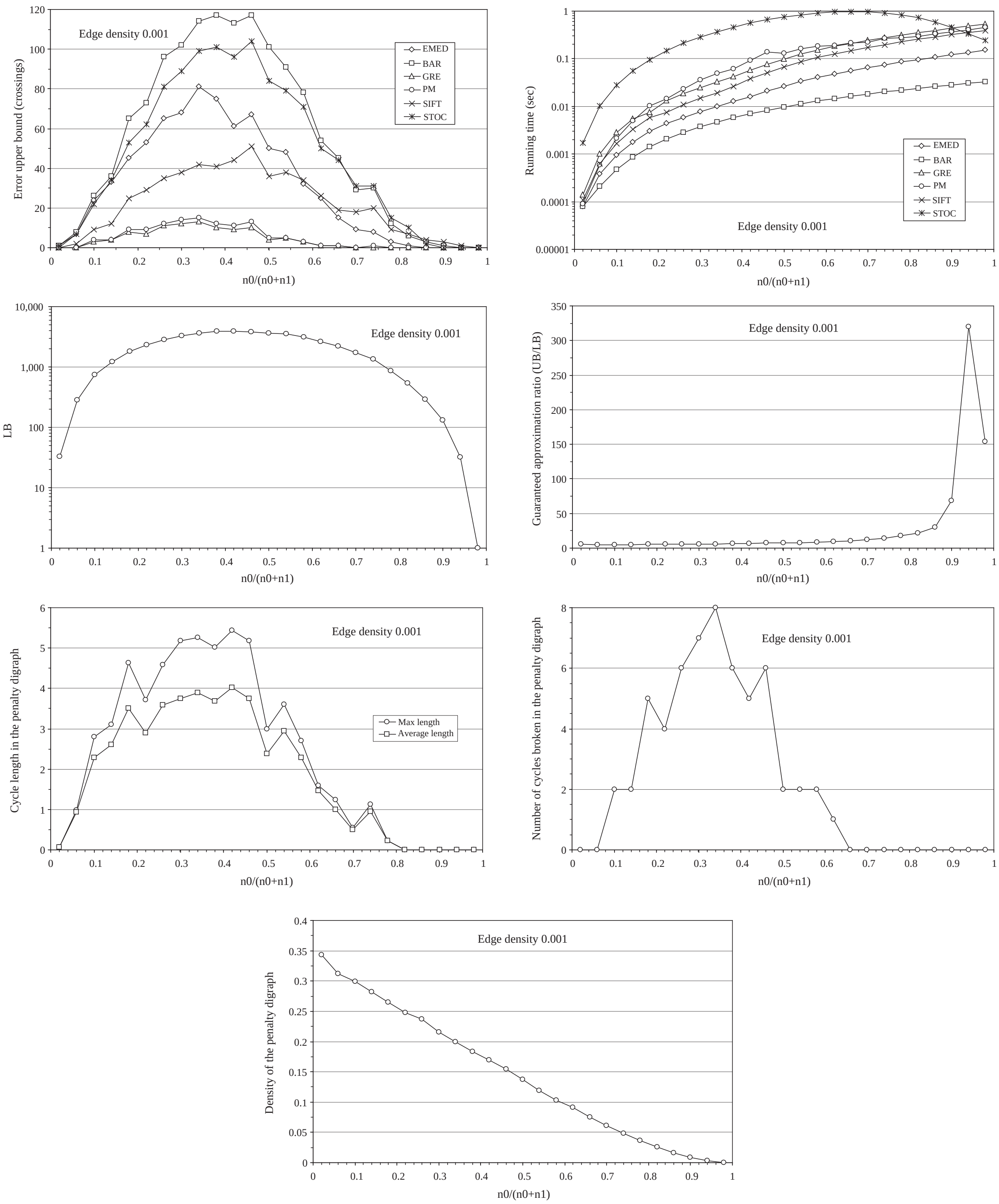


Figure 12: Increasing layers size ratio: $n_0 + n_1 = 1000$, $\rho = 0.001$, and $0.02 \leq \frac{n_0}{n_0+n_1} \leq 0.98$.

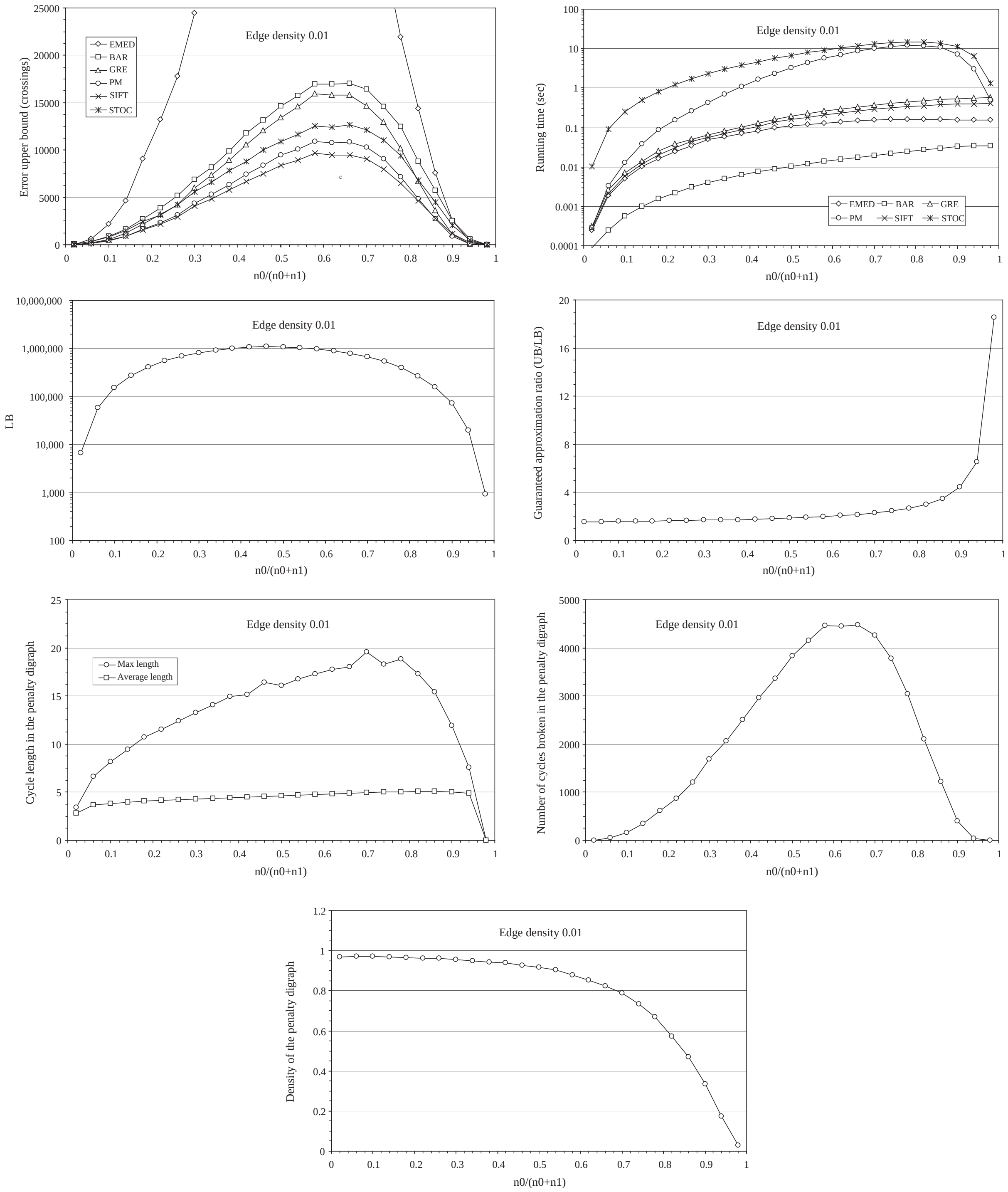


Figure 13: Increasing layers size ratio: $n_0 + n_1 = 1000$, $\rho = 0.01$, and $0.02 \leq \frac{n_0}{n_0+n_1} \leq 0.98$.

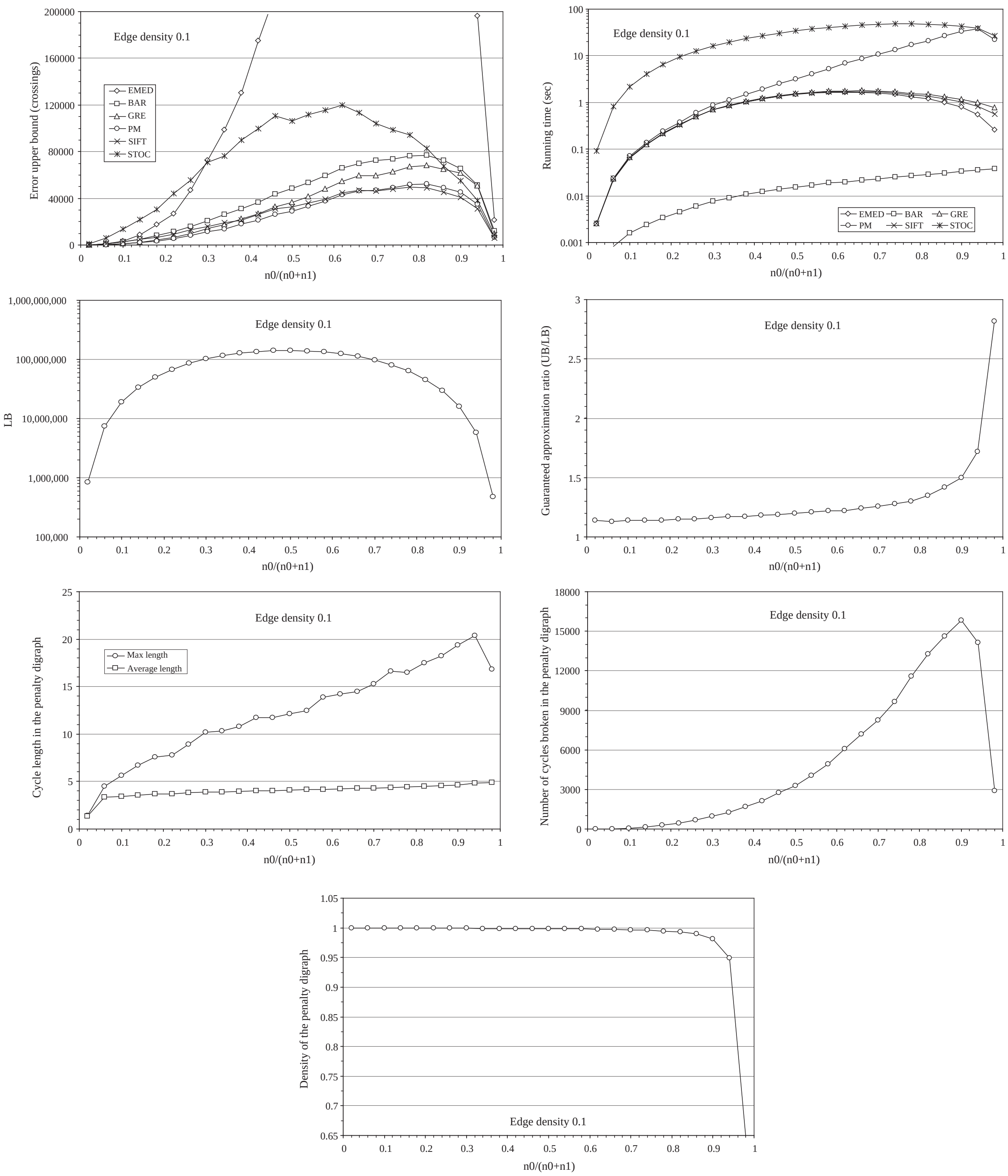


Figure 14: Increasing layers size ratio: $n_0 + n_1 = 1000$, $\rho = 0.1$, and $0.02 \leq \frac{n_0}{n_0+n_1} \leq 0.98$.

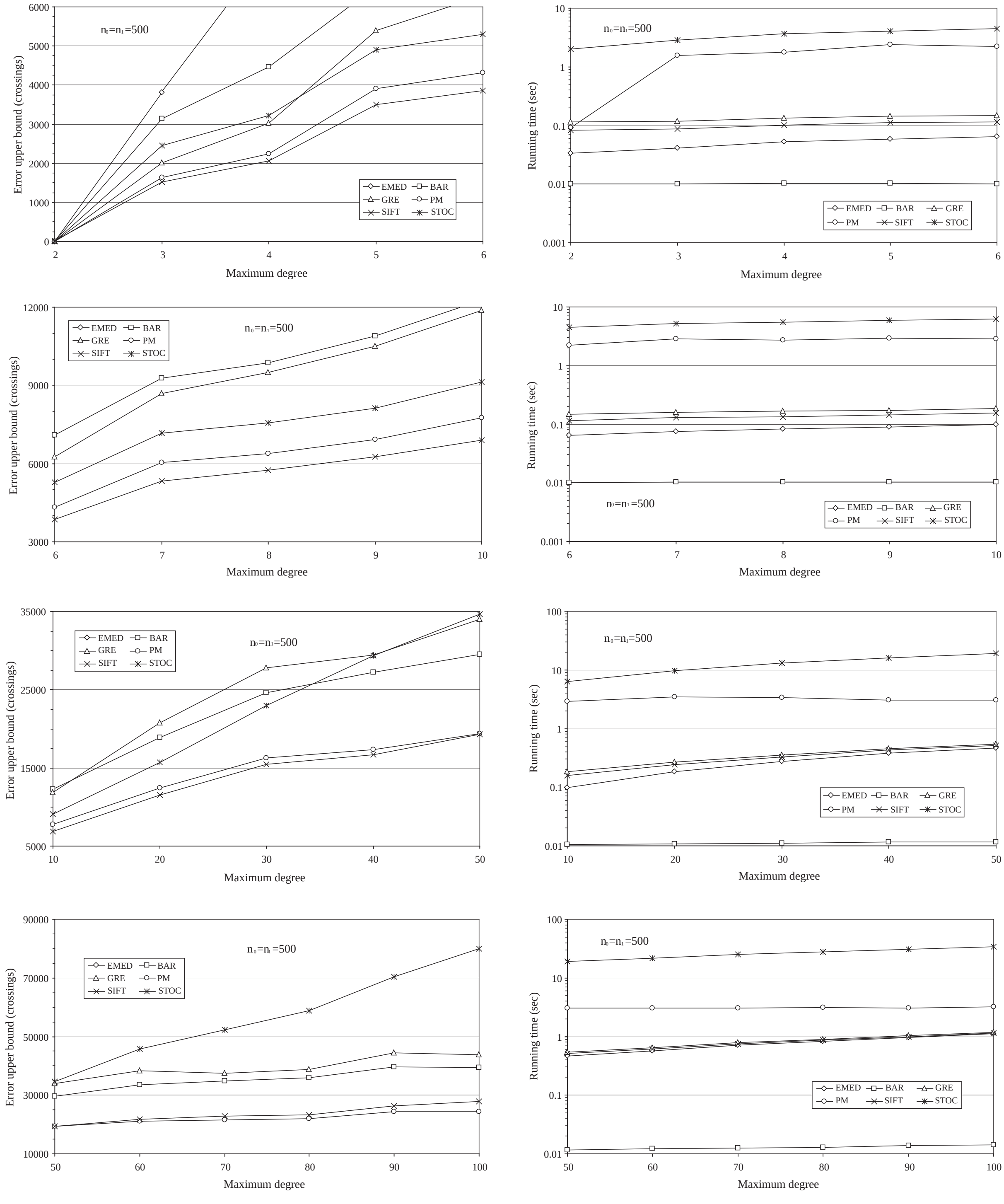


Figure 15: Error upper bound and running time for increasing maximum degree: $n_0 = n_1 = 500$ and $1 \leq d \leq 100$. The charts on each row show the results for different maximum degree intervals.

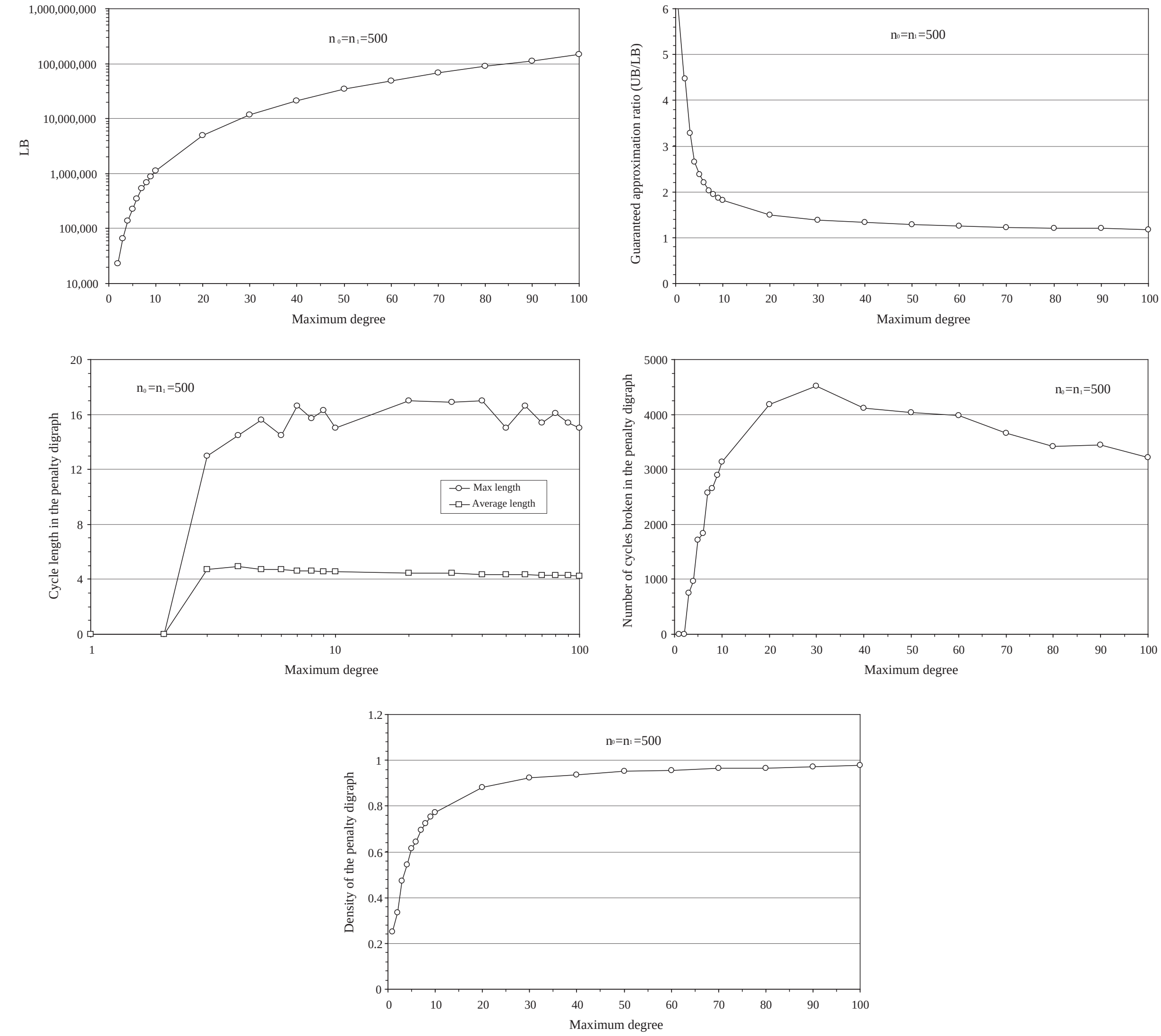


Figure 16: Structural measures for increasing maximum degree: $n_0 = n_1 = 500$ and $1 \leq d \leq 100$.

5.1.4 Increasing maximum degree

In this experiment we study how the performances of the algorithms depend on maximum degree d of vertices of the bipartite graphs. This test is interesting not only because the lower is the maximum degree, the better is the theoretical performance of the greedy algorithm, but also because limited degree graphs frequently arise in practical applications, as already observed in [22]. Graph instances for this experiment are generated using MD; the total number of vertices is fixed to 1000, the layers have the same size, and d increases from 2 to $100 = \frac{n}{10}$. For clarity, Figure 15 reports the results related to the error upper bound and to the running time divided into four different intervals for d.

Though one could expect GRE to find the best solutions when d is very small, this assumption is disproved: actually, SIFT and PM insist at being the best algorithms in the ranges $[2, 50]$ and $[50, 100]$, respectively, and even STOC beats GRE for d in the range 5 to 30. Nevertheless, according to the theoretical analysis of GRE [34], this algorithm is clearly badly affected by large maximum degrees, being outperformed even by BAR when $d > 10$.

					Error Upper Bound					
k	n1	n0	LB	UB/LB	MED	BAR	GRE	PM	SIFT	STOC
3	3	7	8	3.89	0	0	0	0	0	0
4	4	15	95	2.77	0	0	0	0	0	0
5	5	31	756	2.28	5	2	0	0	0	0
6	6	63	4998	2.03	62	21	6	9	4	8
7	7	127	29745	1.87	435	130	85	55	43	96
8	8	255	165375	1.76	2785	713	702	330	295	449
9	9	511	876756	1.69	15523	3510	3877	1924	1595	2658
10	10	1023	4487493	1.63	81245	16346	19310	9538	8297	12364
11	11	2047	22352974	1.58	407538	73618	88517	46033	40327	59837

	Measures on the Penalty Digraph				Running Time (sec)					
k	Density	#Cycle	MaxCyLen	AvgCyLen	MED	BAR	GRE	PM	SIFT	STOC
3	0.8571	0	0	0	0.00002	0.00001	0.00003	0.00002	0.00002	0.00016
4	0.8571	0	0	0	0.00006	0.00002	0.00009	0.00007	0.00008	0.00100
5	0.8645	0	0	0	0.00021	0.00005	0.00040	0.00024	0.00028	0.00556
6	0.8827	3	4	4	0.00085	0.00018	0.00164	0.00118	0.00120	0.03250
7	0.8994	25	6	4.12	0.00379	0.00071	0.00769	0.01429	0.00588	0.18000
8	0.9147	149	10	4.30	0.02000	0.00270	0.03333	0.08000	0.02200	1.00000
9	0.9272	831	17	4.65	0.12000	0.01000	0.23000	1.47000	0.18000	6.66000
10	0.9375	3967	24	4.95	0.64000	0.04333	1.29000	23.53000	1.01000	45.66000
11	0.9458	18319	30	5.24	3.14000	0.16000	5.72000	265.49000	4.52000	322.52000

Figure 17: Performances of algorithms on Warfield instances with $3 \leq k \leq 11$.

Running times show the usual scenario, that regards STOC and PM as the slowest algorithms. However, notice that the running time of PM remains almost constant (around 2 seconds) on all the instances, and differs from the time required by MED, SIFT, and GRE at most by an order on the sparsest instances. On the contrary, STOC running time increases up to reach 35 seconds.

Due to the way generator MD is conceived, increasing d implies increasing the edge density of the bipartite graph, yet at a slower rate. We therefore refer to the experiment presented in Section 5.1.1 for remarks useful to explain the trend of the curves concerning structural measures.

5.2 Warfield instances

The algorithms were tested on Warfield instances up to 2058 vertices ($3 \leq k \leq 11$). The results are reported in the tables in Figure 17. SIFT behaves always better than the other algorithms, yet requiring an amount of time considerably smaller than PM and STOC on the biggest instances ($k = 10, 11$). Surprisingly, on Warfield instances with $k > 8$, BAR finds solutions with less crossings than GRE. We remark that on Warfield instances both LB and the error upper bound for any algorithm exhibit an exponential growth. Moreover, it is interesting to observe that the average cycle length (AvgCyLen) remains almost constant and the maximum cycle length (MaxCyLen) grows linearly with k, even if the number of cycles broken in the penalty digraph (#Cycle) grows exponentially.

5.3 Solution structure

We performed experiments on instances with solution structure $opt(G, x_1) = LB(G, x_1)$ with the aim of identifying counterexamples where algorithms do not return the optimum solution even if it can be polynomially obtained. Figure 18 shows that BAR, SIFT, STOC, and the basic medians algorithm BMED do not return the optimum solution on the test set introduced in Figure 6(b). An example for which also the enhanced median heuristic MED fails is provided by the Warfield instance for $k = 5$, as reported in Figure 17. We found no counterexample for GRE. For what concerns PM, it always finds an optimum solution as already stated in Section 2.2.

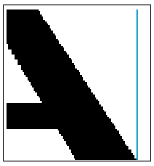
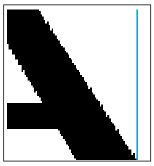
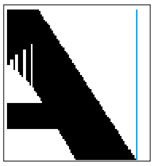
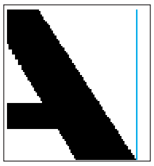
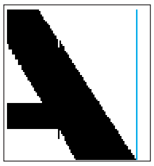
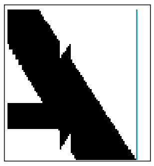
	Solution Structure	Scrambled Solution	BMED Output	BAR Output	MED, GRE, PM Output	SIFT Output	STOC Output
Structure Plot							
Err	0	1464729	287	3209	0	411	7911

Figure 18: Experiment on a test set with solution structure $opt(G, x_1) = LB(G, x_1)$.

5.4 Real test sets

Experiments on real test sets were targeted at confirming or disproving conclusions drawn on synthetic instances about the relative performances of algorithms. Differently from random test sets, for which we averaged the values of the performance indicators on many trials, numerical results on bipartite graphs from real applications are reported *verbatim*.

We also empirically analyzed in the software visualization tool Leonardo the visual complexity of many drawings produced by the algorithms. In this paper we report only a few of them. In particular, Figure 19 shows hierarchical drawings generated with different 2-layer crossing minimization subroutines for a connectivity graph of some Europe cities (Europe Tour) provided as a test example in Leonardo's algorithm repository. Figure 20 shows hierarchical drawings of the digraph dg_11 taken from the e-mail graph service at Bell Labs. For what concerns bipartite drawings, Figure 21 illustrates the results of various algorithms and the corresponding structure plots for the bipartite graph extracted from the NIST matrix BCSPWR01.

All these examples visually confirm the utility of crossing minimization with respect to any random solution, but suggest that the differences on the number of crossings which numerically separate the behavior of different algorithms are extremely difficult to perceive by human observers. Besides, drawings with more crossings may appear more readable simply because of other properties, such as symmetry, as reported in Figure 21. For this reason, variants of the one-sided crossing minimization problem which take into account also other aesthetic criteria such as symmetry, edge length, and angular resolution should be studied.

Figures 22, 23, 24, 25, and 26 report structure plots and histograms concerning both error upper bound and running time for several real test sets from NIST and from digitized images. The results for these instances generally agree with the charts related to random test sets if compared according to size, density, and layer size ratio, corroborating the conclusions drawn from experiments on synthetic test sets.

6 Conclusions

We considered the one-sided crossing minimization problem: given a bipartite graph G and a permutation x_0 of the vertices on a layer, find a permutation x_1 of the vertices on the other layer which minimizes the number of edge crossings in any bipartite straightline drawing of G where vertices are sorted according to x_0 and x_1. Several algorithms for solving this problem have been proposed in the literature due to its relevance for producing layered drawings of directed graphs. A new approximation algorithm devised in this paper, based on the penalty minimization technique, is another useful, practical alternative according to the experimental results.

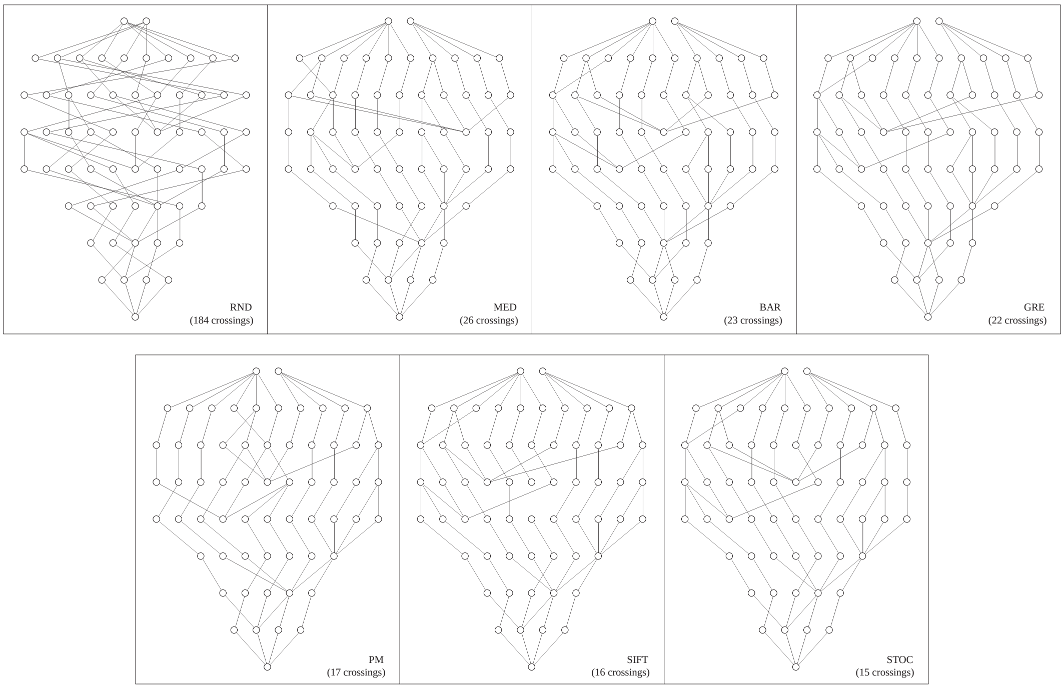


Figure 19: Hierarchical drawings of the Europe Tour graph generated by the algorithm visualization system Leonardo with different one-sided crossing minimization subroutines.

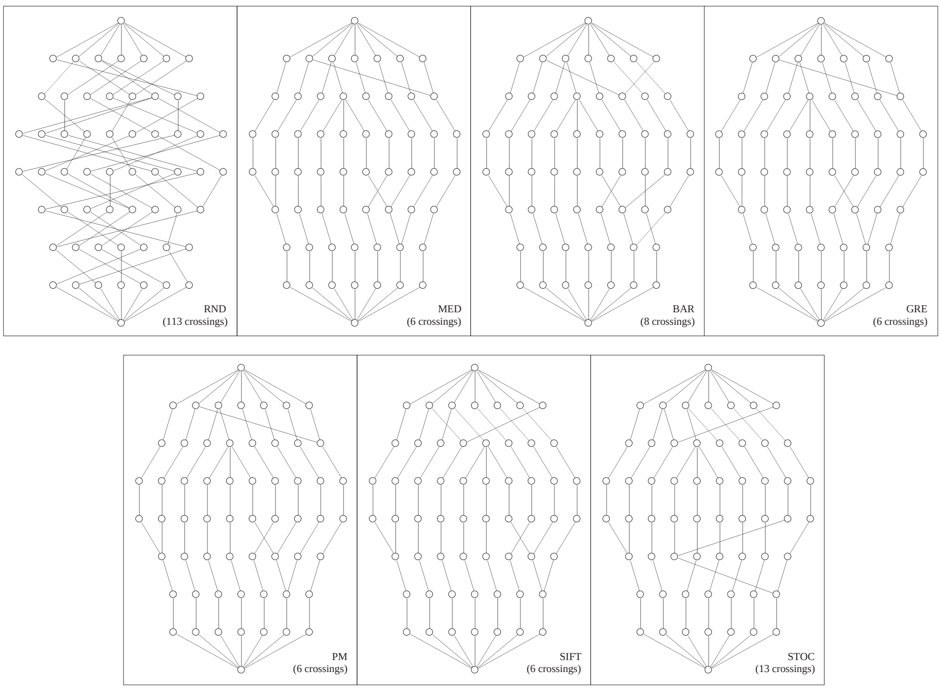
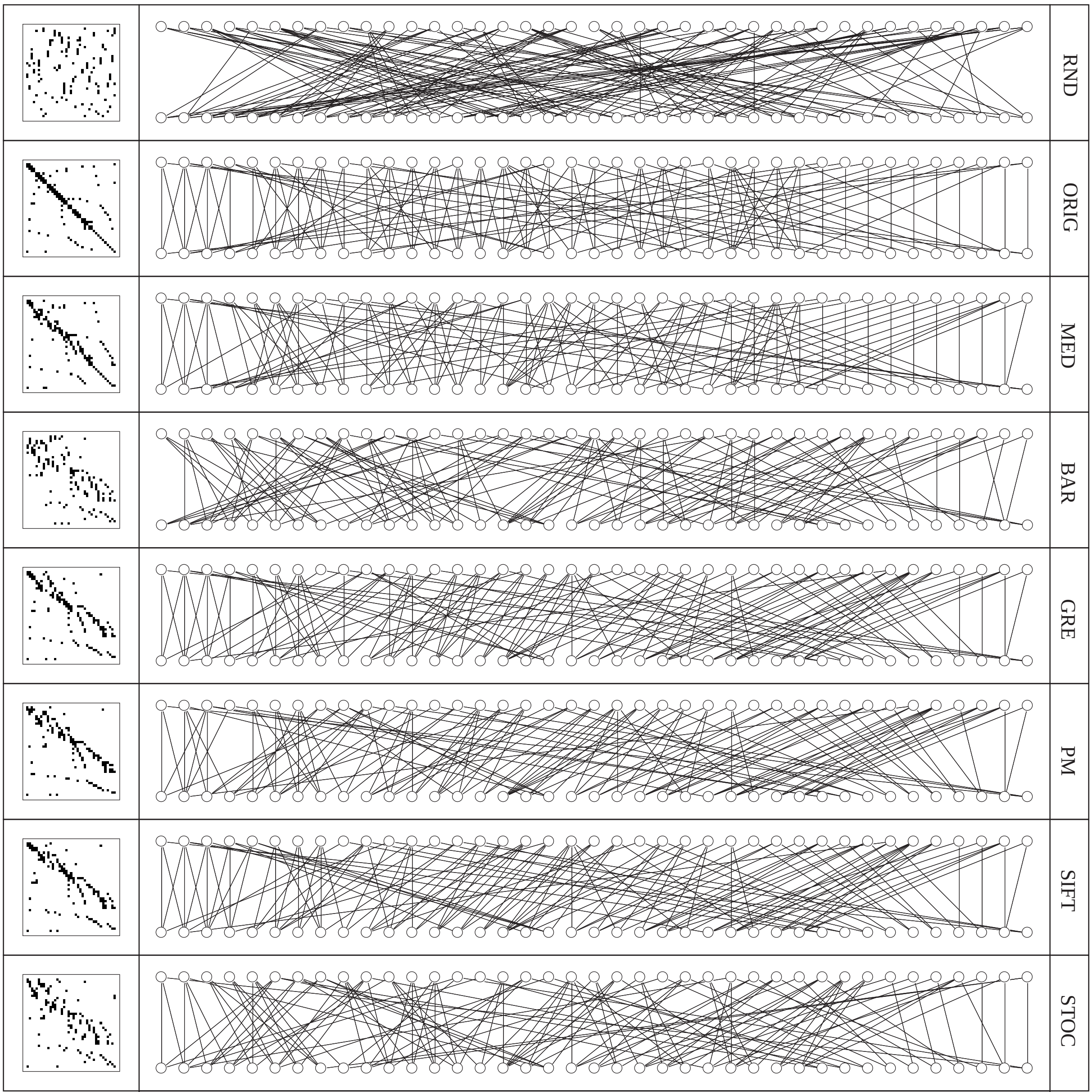


Figure 20: Hierarchical drawings of the digraph dg_11 from the e-mail graph drawing service at Bell Labs generated by the algorithm visualization system Leonardo with different one-sided crossing minimization subroutines.



	LB	UB	ORIG	RND	MED	BAR	GRE	PM	SIFT	STOC
Crossings	1725	6442	1926	3708	1791	1888	1732	1732	1734	1869
Crossings-LB	0	4717	201	1983	66	163	7	7	9	144
Running Time	-	-	-	-	0.4	0.1	0.6	0.6	0.5	13.3

UB/LB
3.7

Figure 21: Different drawings and corresponding structure plots of the bipartite graph extracted from the NIST matrix BCSPWR01 ($\rho = 0.0861$): drawing ORIG corresponds to the original structure plot and appears more readable than the drawings produced by the algorithms due to its symmetry properties, even if it has more crossings.

We empirically compared the performance of PM with the performance of several previous algorithms. Experiments were performed on real test sets, on pathological instances for CP, and on various kinds of randomly generated bipartite graphs for many different settings of the parameters determining size and structure of the problem instances. Among the performance indicators, we considered not only the number of edge crossings and the running time, but also structural measures of the problem instances and of the penalty digraph.

In spite of the combinatorial straitjacket of CP due to the high values of LB, algorithms clearly exhibit different behaviors as revealed by experiments both on synthetic and on real test sets, but from the point of view of visual complexity, we found no clear winner: the differences in terms of number of crossings between the drawings produced by different algorithms are usually hard to tell by human observers, especially on dense graphs. However, besides the fact that graphs arising in most real applications are very sparse, the visual comparison between the drawings produced by the considered algorithms for CP and the layouts obtained with no optimization attempts clearly proves that trying to minimize edge crossings is very useful.

We also observed that drawings with more crossings may appear more readable simply because of other aesthetic criteria, such as symmetry, edge length, and angular resolution. For this reason, we believe that variants of the one-sided crossing minimization problem taking into account also these criteria should be studied.

The interested reader can find a web page concerning the full package used in the preparation of this paper at the URL `http://www.dsi.uniroma1.it/~finocchi/experim/cplib-2.0/`.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Approximate Solution of NP-Hard Optimization Problems*. Springer Verlag, 1999.

[2] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. In *Proc. 1st Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'98)*, pages 49–62, 1998.

[3] R. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. Dongarra. Matrix Market: a web resource for test matrix collections. In R. Boisvert, editor, *The Quality of Numerical Software: Assessment and Enhancement*, pages 125–137. Chapman and Hall, London, 1997. Matrix Market is available at the URL: `http://math.nist.gov/MatrixMarket/`.

[4] M.J. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Trans. Syst. Man Cybern.*, 10(11):705–715, 1980.

[5] T. Catarci. The assignment heuristic for crossing reduction. *IEEE Trans. Syst. Man Cybern.*, 25(3):515–521, 1995.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[7] P. Crescenzi, C. Demetrescu, I. Finocchi, and R. Petreschi. Reversible execution and visualization of programs with Leonardo. *Journal of Visual Languages and Computing*, 11(2), 2000. Leonardo is available at the URL: `http://www.dis.uniroma1.it/~demetres/Leonardo/`.

[8] C. Demetrescu and I. Finocchi. Leonardo's repository of animated algorithms. Available at the URL: `http://www.dis.uniroma1.it/~demetres/Leonardo/ProgramsRepository.html`.

[9] C. Demetrescu and I. Finocchi. A note on feedback problems on directed graphs. Technical Report TR-35-99, University of Rome "La Sapienza", Department of Computer and Systems Science, 1999.

[10] Demetrescu, C. and Finocchi, I. Break the "right" cycles and get the "best" drawing. In B.E. Moret and A.V. Goldberg, editors, *Proc. of the 2nd International Workshop on Algorithm Engineering and Experiments (ALENEX'00)*, pages 171–182, 2000.

[11] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River, NJ, 1999.

[12] S. Dresbach. A new heuristic layout algorithm for dags. In U. Derigs and A. Drexl, editors, *Operations Research Proceedings*, pages 121–126, 1994.

[13] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combin.*, 21:89–98, 1986.

[14] P. Eades and X. Lin. A new heuristic for the feedback arc set problem. *Australian Journal of Combinatorics*, 12:15–26, 1995.

[15] P. Eades, X. Lin, and W.F. Smyth. A Fast and Effective Heuristic for the Feedback Arc Set Problem. *Information Processing Letters*, 47(6):319–323, 1993.

[16] P. Eades and S. Whitesides. Drawings graphs in two layers. *Theoretical Computer Science*, 131:361–374, 1994.

[17] P. Eades and N.C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

[18] G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics. In *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 62–71, 1995.

[19] E.R. Gansner, E. Koutsofios, S.C. North, and K.P. Vo. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 19:214–230, 1993.

[20] M.R.. Garey and D.S. Johnson. *Computers and Intractability: a Guide to Theory of NP-completeness*. W.H.Freeman, 1979.

[21] A.V. Goldberg. Selecting problems for algorithm evaluation. In *Proc. 3-rd Workshop on Algorithm Engineering (WAE'99), LNCS 1668*, pages 1–11, 1999.

[22] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms and Applications*, 1(1):1–25, 1997.

[23] D.E. Knuth. *The Stanford GraphBase: A platform for combinatorial computing*. Addison-Wesley, 1993.

[24] E. Makinen. Experiments in drawing 2-level hierarchical graphs. Technical Report A-1988-1, Department of Computer Science, University of Tampere, 1988.

[25] C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. In *Proc. 7th International Symposium on Graph Drawing (GD'99)*, LNCS 1731, pages 217–224, 1999.

[26] P. Mutzel, C. Gutwenger, R. Brockenauer, and S. Fialko. A library of algorithms for graph drawing. *Lecture Notes in Computer Science*, 1547:456–457, 1998. Project URL: `http://www.mpi-sb.mpg.de/AGD/`.

[27] S. North. 5114 directed graphs. Available at `ftp://ftp.research.att.com/dist/drawdag/`, 1995.

[28] H.C. Purchase, R.F. Cohen, and M.I. James. An experimental study of the basis for graph drawing algorithms. *ACM Journal on Experimental Algorithmics*, 2, 1997.

[29] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. International Conference on Computer-Aided Design (ICCAD'93)*, pages 42–47, 1993.

[30] P.D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15:281–288, 1995.

[31] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981.

[32] V. Valls, R. Marti, and P. Lino. A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs. *J. Operational Research*, 90:303–319, 1996.

[33] J.N. Warfield. Crossing theory and hierarchy mapping. *IEEE Trans. Syst. Man Cybern.*, 7(7):502–523, 1977.

[34] A. Yamaguchi and A. Sugimoto. An approximation algorithm for the two-layered graph drawing problem. In *Proc. 5th Annual Int. Conference on Computing and Combinatorics (COCOON'99)*, pages 81–91, 1999.

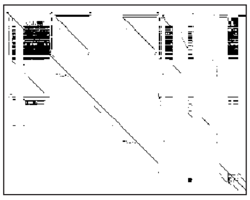
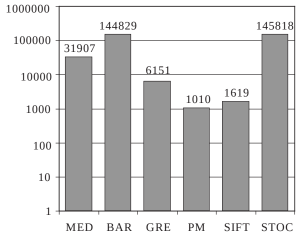
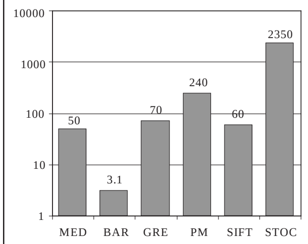
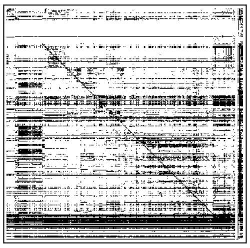
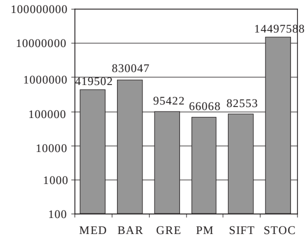
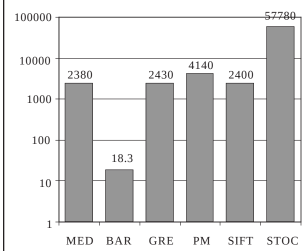
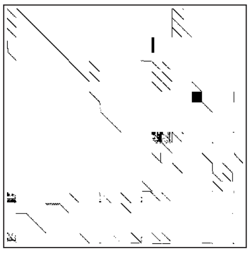
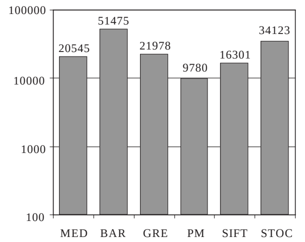
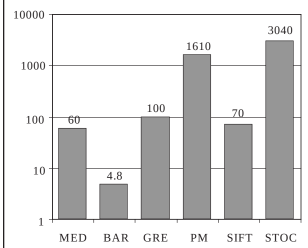
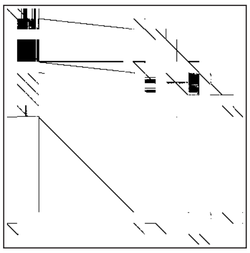
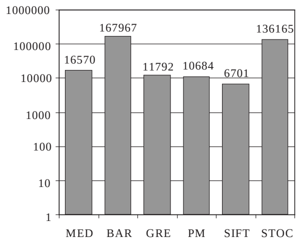
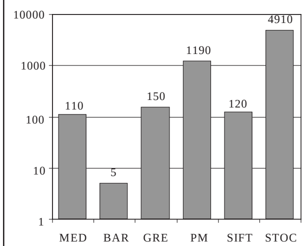
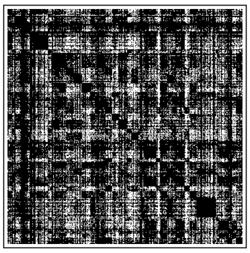
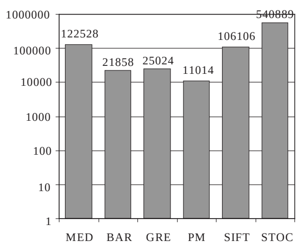
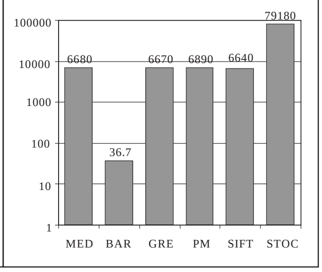
Matrix Information	Structure Plot	Error Upper Bound	Running Time (msec)
Name: WM1 **Discipline:** Economic Models **Description:** One of fifteen regions in world economic model **Set:** ECONIEA **Collection:** Harwell-Boeing **Accession:** Summer, 1982 **Size:** 207x277 **Density:** 0.0507 **Layers ratio:** 0.5723 **UB/LB:** 1.53			
Name: BEAUSE **Discipline:** Economic Models **Description:** US economic transactions in 1972: commodities x industries **Set:** ECONIEA **Collection:** Harwell-Boeing **Accession:** Summer, 1982 **Size:** 479x507 **Density:** 0.1768 **Layers ratio:** 0.5 **UB/LB:** 1.3			
Name: MAHINDAS **Discipline:** Economic models **Description:** Economic model of Victoria, Australia, 1880 **Set:** ECONAUS **Collection:** Harwell-Boeing **Accession:** October 24, 1984 **Source:** Ken Pearson, Dept. of Mathematics, La Trobe Univ., Melbourne, Au. **Size:** 351x351 **Density:** 0.0185 **Layers ratio:** 0.5 **UB/LB:** 2.48			
Name: ORANI768 **Discipline:** Economic models **Description:** Economic model of Australia, 1968-69 **Set:** ECONAUS **Collection:** Harwell-Boeing **Accession:** October 24, 1984 **Source:** Ken Pearson, Dept. of Mathematics, La Trobe Univ., Melbourne, Au. **Size:** 351x351 **Density:** 0.0364 **Layers ratio:** 0.5 **UB/LB:** 2.62			
Name: PSMIGR1 **Discipline:** Demography **Description:** US inter-county migration 1965-1970 **Set:** PSMIGR **Collection:** Harwell-Boeing **Accession:** November 1983 **Source:** Paul Slater, University of California, Santa Barbara **Size:** 351x351 **Density:** 0.7376 **Layers ratio:** 0.5 **UB/LB:** 1.05			

Figure 22: Real test sets from NIST and experimental results: economic and demographic models.

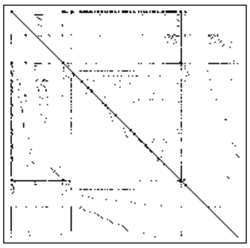
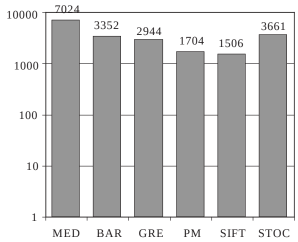
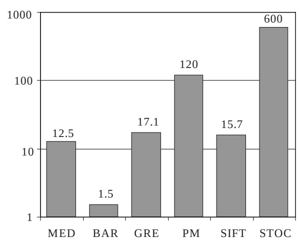
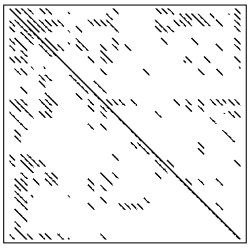
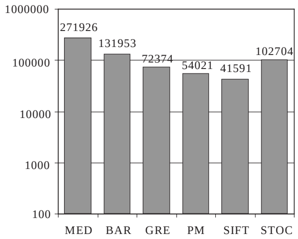
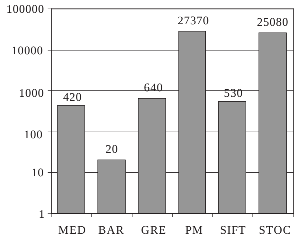
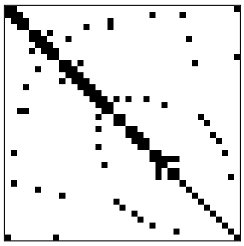
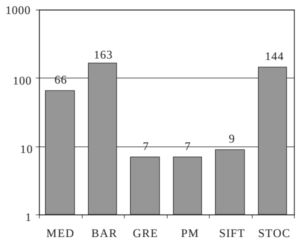
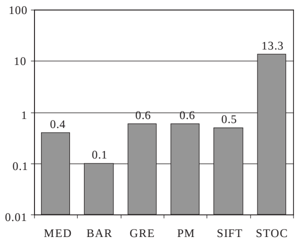
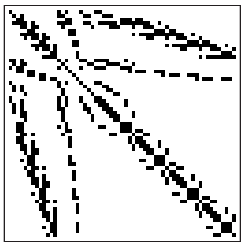
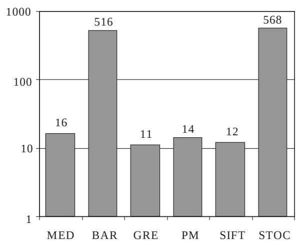
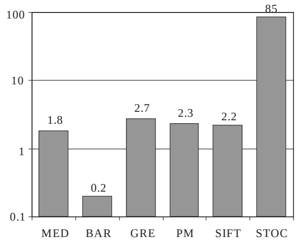
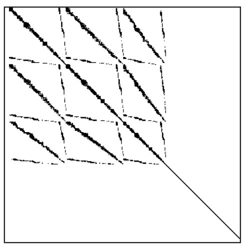
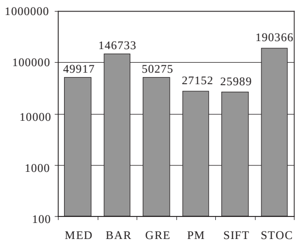
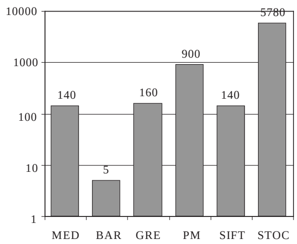
Matrix Information	Structure Plot	Error Upper Bound	Running Time (msec)
Name: FS183_3 **Discipline:** Chemical Kinetics **Description:** PSMOG atmospheric polution study **Set:** FACSIMILE **Collection:** Harwell-Boeing **Accession:** August, 1983 **Source:** Alan Curtis, Computer Science and Systems Division, AERE Harwell. **Size:** 183x183 **Density:** 0.0319 **Layers ratio:** 0.5 **UB/LB:** 1.83			
Name: FS760_1 **Discipline:** Chemical Kinetics **Description:** STRAT stratospheric ionization study **Set:** FACSIMILE **Collection:** Harwell-Boeing **Accession:** August, 1983 **Source:** Alan Curtis, Computer Science and Systems Division, AERE Harwell. **Size:** 760x760 **Density:** 0.0103 **Layers ratio:** 0.5 **UB/LB:** 1.66			
Name: BCSPWR01 **Discipline:** Power network patterns **Description:** Standard IEEE test power system -- New England **Set:** BCSPWR **Collection:** Harwell-Boeing **Accession:** October 24, 1984 **Source:** B. Dembart and J. Lewis, Boeing Comp. Services, Seattle, WA, USA **Size:** 39x39 **Density:** 0.0861 **Layers ratio:** 0.5 **UB/LB:** 3.73			
Name: CAN61 **Discipline:** Aircraft design **Description:** Structures problems in aircraft design **Set:** CANNES **Collection:** Harwell-Boeing **Accession:** June 3, 1981 **Source:** Lucien Marro, Programm. Scientifique, Aerospatiale Cannes, France **Size:** 61x61 **Density:** 0.1497 **Layers ratio:** 0.5 **UB/LB:** 2.01			
Name: ZENIOS **Discipline:** Air-traffic Control **Description:** Air-traffic Control Model **Set:** AIRTFC **Collection:** Harwell-Boeing **Accession:** Autumn 1985 **Source:** Stavros Zenios, Princeton University **Size:** 351x351 **Density:** 0.0450 **Layers ratio:** 0.5 **UB/LB:** 1.8			

Figure 23: Real test sets from NIST and experimental results: miscellaneous applications.

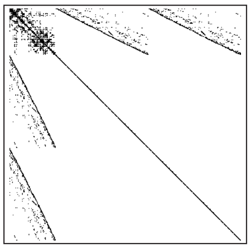
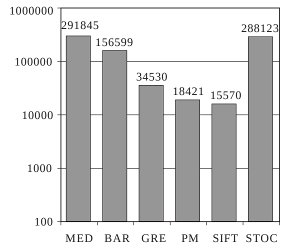
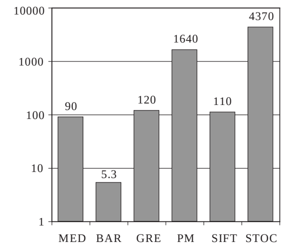
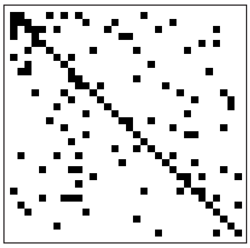
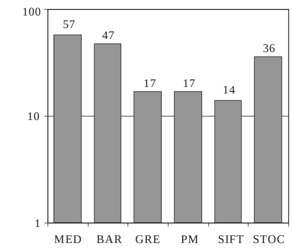
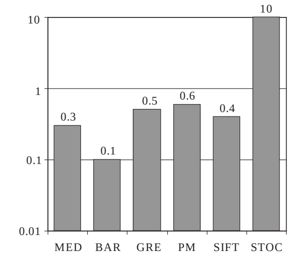
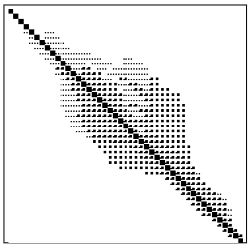
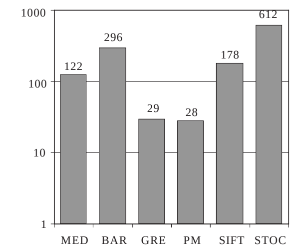
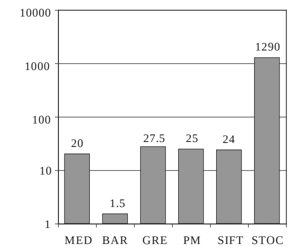
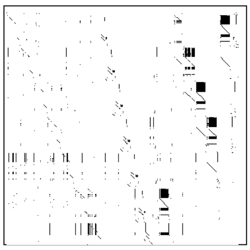
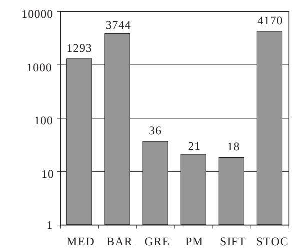
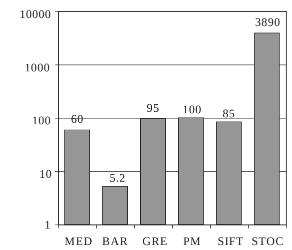
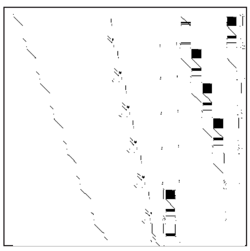
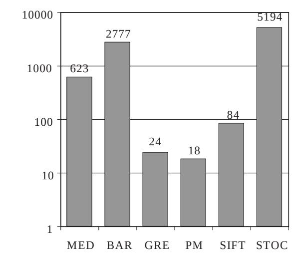
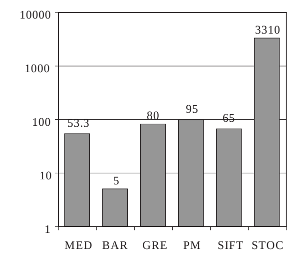
Matrix Information	Structure Plot	Error Upper Bound	Running Time (msec)
Name: ADD20 **Discipline:** Electronic circuit design **Description:** 20 bit adder **Set:** HAMM **Collection:** Independent Sets and Generators **Accession:** Fall 1995 **Source:** Steve Hamm, Motorola Inc. Semiconductor Systems Design Techn. **Size:** 351x351 **Density:** 0.0310 **Layers ratio:** 0.5 **UB/LB:** 1.72			
Name: IBM32 **Discipline:** Variety of disciplines **Description:** Original Harwell sparse matrix test collection 1971 IBM conference advertisement **Set:** SMTAPE **Collection:** Harwell-Boeing **Accession:** Summer 1978 **Source:** Harwell Lab, England **Size:** 32x32 **Density:** 0.1230 **Layers ratio:** 0.5 **UB/LB:** 2.04			
Name: MCCA **Discipline:** Astrophysics **Description:** Radiative transfer and statistical equilibrium **Set:** ASTROPH **Collection:** Harwell-Boeing **Accession:** March, 1985 **Source:** Mats Carlsson, Institute of Theoretical Astrophysics, University of Oslo **Size:** 180x180 **Density:** 0.0821 **Layers ratio:** 0.5 **UB/LB:** 5.46			
Name: STR 200 **Discipline:** Variety of disciplines **Description:** Simplex method basis matrix **Set:** SMTAPE **Collection:** Harwell-Boeing **Accession:** Summer 1978 **Source:** Harwell Laboratory, England **Size:** 363x363 **Density:** 0.0233 **Layers ratio:** 0.5 **UB/LB:** 13.93			
Name: STR 0 **Discipline:** Variety of disciplines **Description:** Simplex method basis matrix **Set:** SMTAPE **Collection:** Harwell-Boeing **Accession:** Summer 1978 **Source:** Harwell Laboratory, England **Size:** 363x363 **Density:** 0.0186 **Layers ratio:** 0.5 **UB/LB:** 12.61			

Figure 24: Real test sets from NIST and experimental results: miscellaneous applications.

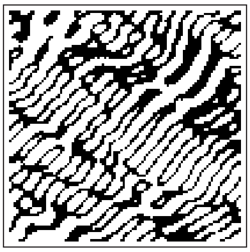
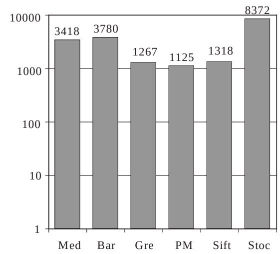
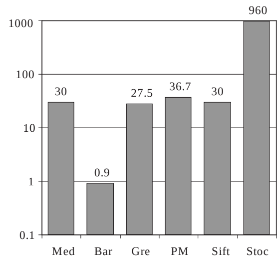
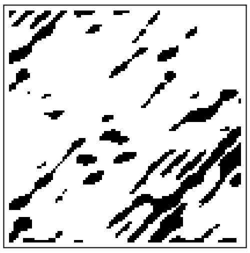
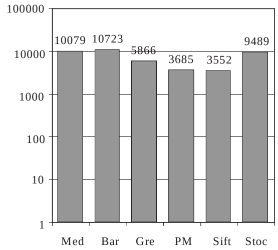
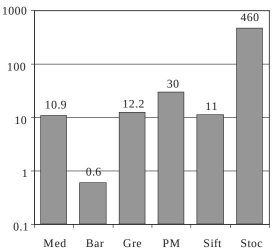
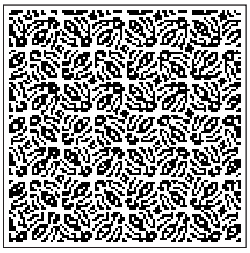
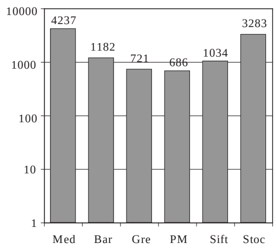
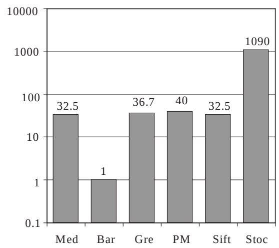
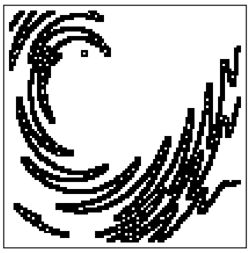
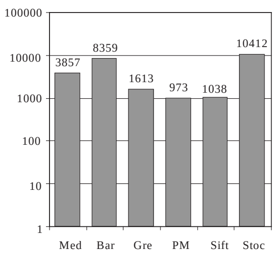
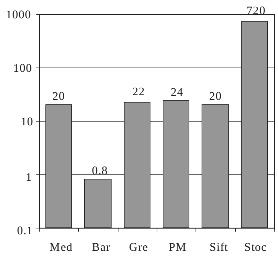
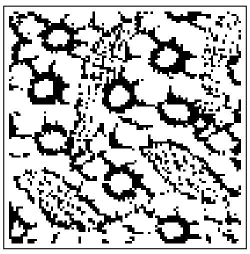
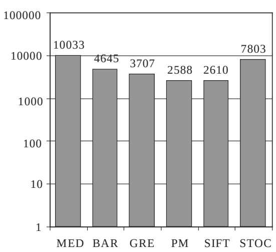
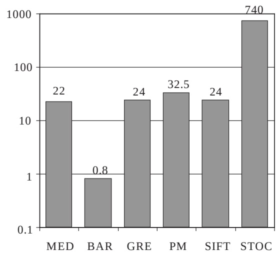
Matrix Information	Structure Plot	Error Upper Bound	Running Time (msec)
Name: TEXTURE1 **Description:** Bitmapped image **Size:** 100x100 **Density:** 0.3981 **Layers ratio:** 0.5 **UB/LB:** 1.22			
Name: TEXTURE2 **Description:** Bitmapped image **Size:** 100x100 **Density:** 0.1803 **Layers ratio:** 0.5 **UB/LB:** 1.58			
Name: TEXTURE3 **Description:** Bitmapped image **Size:** 100x100 **Density:** 0.4532 **Layers ratio:** 0.5 **UB/LB:** 1.08			
Name: TEXTURE4 **Description:** Bitmapped image **Size:** 100x100 **Density:** 0.2934 **Layers ratio:** 0.5 **UB/LB:** 2.58			
Name: TEXTURE5 **Description:** Bitmapped image **Size:** 100x100 **Density:** 0.2995 **Layers ratio:** 0.5 **UB/LB:** 1.28			

Figure 25: Real test sets: digitized textures.

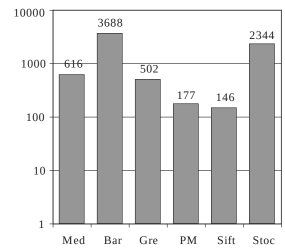
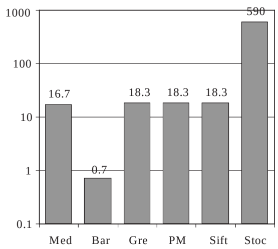
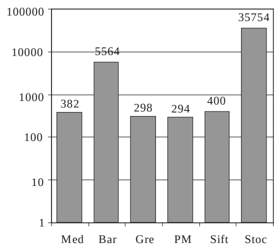
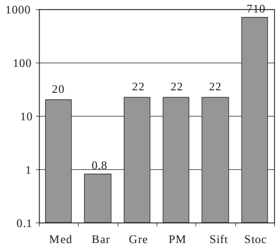
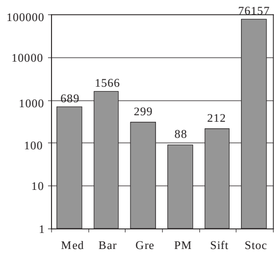
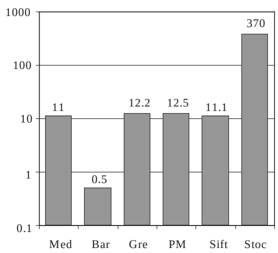
Matrix Information	Structure Plot	Error Upper Bound	Running Time (msec)
Name: WARHOL **Description:** Bitmapped image **Size:** 100x65 **Density:** 0.5951 **Layers ratio:** 0.3939 **UB/LB:** 1.14			
Name: MONALISA **Description:** Bitmapped image **Size:** 100x65 **Density:** 0.7155 **Layers ratio:** 0.3939 **UB/LB:** 1.2			
Name: LEONARDO **Description:** Bitmapped image **Size:** 90x64 **Density:** 0.4193 **Layers ratio:** 0.4156 **UB/LB:** 1.6			

Figure 26: Real test sets: digitized paintings and images.