Journal of Discrete Algorithms 7 (2009) 439–452

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Crossing minimization in weighted bipartite graphs [☆]

Olca A. Çakıroğlu [1], Cesim Erten [1], Ömer Karataş [*,1], Melih Sözdinler

Computer Science and Engineering, Işık University Şile, Istanbul 34980, Turkey

ARTICLE INFO

Article history:
Received 4 September 2007
Received in revised form 10 March 2008
Accepted 4 August 2008
Available online 3 September 2008

Keywords:
Weighted bipartite graphs
Crossing minimization
Combinatorial optimization
Approximation algorithms

ABSTRACT

Given a bipartite graph $G = (L_0, L_1, E)$ and a fixed ordering of the nodes in L_0, the problem of finding an ordering of the nodes in L_1 that minimizes the number of crossings has received much attention in literature. The problem is NP-complete in general and several practically efficient heuristics and polynomial-time algorithms with a constant approximation ratio have been suggested. We generalize the problem and consider the version where the edges have nonnegative weights. Although this problem is more general and finds specific applications in automatic graph layout problems similar to those of the unweighted case, it has not received as much attention. We provide a new technique that efficiently approximates a solution to this more general problem within a constant approximation ratio of 3. In addition we provide appropriate generalizations of some common heuristics usually employed for the unweighted case and compare their performances.

1. Introduction

Let $G = (L_0, L_1, E)$ be a bipartite graph where L_0, L_1 indicate the two layers of the graph and E denotes the edge set. The crossing minimization problem consists of finding an ordering of the nodes in L_0 and L_1 such that placing the two layers on two horizontal lines and drawing each edge as a straight line segment, the number of pairwise edge crossings is minimized. A related version is one where the ordering in one of the layers is already fixed. The former is usually referred to as the *both layers free bipartite crossing minimization* whereas the latter as the *one layer free bipartite crossing minimization*. Both problems have been extensively studied in literature. Unfortunately they are both NP-hard [7,12]. As a result, extensive research has been devoted to the design of heuristics and approximation algorithms for these problems.

Both crossing minimization problems, especially the one layer free version, have been used as basic building blocks for automatic layout of directed graphs following the approach of Sugiyama, Tagawa, and Toda [21]. This approach consists mainly of three steps: Assigning the nodes to horizontal layers, ordering the nodes within each layer so as to minimize the number of crossings, and finally assigning actual coordinates for the nodes and edge bends. A commonly used approach for the second step is the "layer-by-layer sweep" method which requires a solution to the one layer free crossing minimization problem.

The current paper examines the weighted generalization of the (unweighted) one layer free crossing minimization problem, from now on referred to as OLF. Specifically we consider the following:

[☆] A preliminary version of this paper appeared in Proceedings of Workshop on Experimental Algorithms (WEA'07).
[*] Corresponding author.
E-mail addresses: arda.cakiroglu@isik.edu.tr (O.A. Çakıroğlu), cesim@isikun.edu.tr (C. Erten), omer.karatas@isik.edu.tr (Ö. Karataş), melih@isikun.edu.tr (M. Sözdinler).
[1] Partially supported by TUBITAK—The Scientific and Technological Research Council of Turkey, grant 106E071.

1570-8667/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2008.08.003

Weighted One Layer Free Problem (*WOLF*): Given an edge-weighted bipartite graph $G = (L_0, L_1, E)$ and a fixed ordering of nodes in L_0, find an ordering of nodes in L_1 such that the *total weighted crossings* in the resulting drawing is minimized. If two edges $e_1, e_2 \in E$ cross then this crossing amounts to $W(e_1) \times W(e_2)$ in the total weighted crossings, where $W(e_1), W(e_2)$ denote the nonnegative weights of e_1, e_2 respectively.

Besides the fact that it is a generalization of OLF, its wide range of applications provides further motivation to study WOLF. Similar to OLF, natural applications include those related to computing layered layouts of directed graphs. Many such instances assign an edge weight to indicate its "importance". The goal then is to compute layouts with few crossings between important edges of large weight [3,4,9]. Other applications specific to WOLF include recent problems related to wire crossing minimization in VLSI [19].

1.1. Previous work

Crossing minimization problems in drawings of bipartite graphs have been studied extensively in literature. A common method to solve the both layers free version of the problem is to iteratively apply a solution to OLF, while alternating the fixed layer at each iteration. Therefore considerable attention has been devoted to the OLF problem and its variations [2,5,7,8,13,15,17,18,23].

Since OLF is NP-complete even when the graph instances are sparse [17], much of related research concentrates on the design of heuristics and approximation algorithms. Most popular heuristics include the *barycenter* method of [21] and the *median heuristic* of [7]. Jünger and Mutzel survey various heuristics and experimentally compare their performances [13]. They conclude that the barycenter method yields slightly better results than the median heuristic in practice. On the other hand from a theoretical point of view median heuristic is better. Specifically, they both run in linear time and the median heuristic is a 3-approximation, whereas the approximation ratio of the barycenter method is $\Theta(\sqrt{|L_0|})$ [7]. Yamaguchi and Sugimoto [23] provide a greedy algorithm GRE that has the same approximation ratio of 3 in the worst case and that works well in practice. However the running time of GRE is quadratic. Recently, Nagamochi devised a 1.47-approximation algorithm for OLF [18]. Another promising technique for OLF is the penalty graph approach introduced by Sugiyama et al. [21]. The performance of this method depends on an effective solution to the minimum feedback arc set (FAS) problem which is also NP-complete [11]. Demetrescu and Finocchi experimentally compare the performance of the penalty graph method based on their algorithm for FAS to that of the barycenter, median, and the GRE heuristics [5]. In addition to the mentioned heuristics, approaches based on integer programming formulations that solve the problem exactly have been suggested. Jünger and Mutzel showed that OLF can be formulated as a linear ordering problem which is then solved optimally by employing a branch-and-cut technique [13]. Although directly applicable to WOLF, this exact approach works well mostly for sparse graphs of relatively small size.

We note that previously different versions of WOLF have been considered only as a subproblem for solving problems related to automatic graph layout computations. These applications usually employ a weighted modification of the barycenter method [3,4,9] or a penalty graph based approach [19]. Such practices are based on the presupposition that simple extensions of desirable methods for OLF should also lead to efficient solutions for WOLF. To the contrary, our results indicate this may not necessarily be the case.

1.2. Summary of the main results

To the best of our knowledge this is the first study to consider specifically the WOLF problem and to compare various promising methods. The following summarizes the main contributions:

- We provide an efficient approximation algorithm `3-WOLF` for the WOLF problem. Specifically, algorithm `3-WOLF` 3-approximates WOLF in $O(|E| + |V|\log|V|)$ time, where $V = L_0 \cup L_1$. We note that this is the first polynomial time algorithm to have a constant approximation ratio for the WOLF problem. Although there are several polynomial-time algorithms with constant approximation ratios for solving OLF [7,18,23], it is not obvious how to generalize them to WOLF settings while retaining the same constant approximation ratios.
- We devise weighted modifications of common heuristics that are shown to produce outputs with *high quality* for OLF instances in practical settings. Specifically we present extensions of the barycenter, median, GRE, and the penalty graph methods previously suggested for OLF. We experimentally compare the performances of these methods to that of `3-WOLF`. Our experiments indicate that the output quality of `3-WOLF`, in terms of the total weighted crossings, is better than that of the methods with comparable running times. Besides, it produces outputs with qualities comparable to those of the methods with expensive running time requirements.
- The distinction between efficient methods for the OLF and WOLF problems arises as an interesting result of our studies. A particular instance of this is the barycenter method. Although in terms of the quality of outputs it seems to outperform the methods with comparable running times in the OLF settings, its weighted extension does not share the same performance results in the WOLF settings. To emphasize such distinctions we provide theoretical insight supported by experimental analysis.

2. `3-WOLF`: A 3-approximation algorithm for WOLF

Given a bipartite graph $G = (L_0, L_1, E)$, let n_0, n_1 denote the sizes of the layers L_0, L_1 respectively. Assume the nodes in L_0 are labeled with integers from 1 through n_0. For $x \in L_1$ and $j \leqslant l$, let $W(x)_j^l = \sum_{p=j}^{l} W(x, p)$, where $W(x, p)$ is the weight of the edge (x, p). With this notation we assume that $W(x, p) = 0$ if there is no edge between x and p. For simplicity let $W(x)_1^0 = W(x)_{n_0+1}^{n_0} = 0$. For $u, v \in L_1$, let c_{uv} denote the sum of the weighted crossings between the edges incident on u and v when u is placed to the left of v, that is,

$$c_{uv} = \sum_{p=1}^{n_0-1} W(v, p) \times W(u)_{p+1}^{n_0}.$$

We divide c_{uv} into three terms X_{uv}, Y_{uv}, Z_{uv} where $c_{uv} = X_{uv} + Y_{uv} + Z_{uv}$. For a given integer r, where $1 \leqslant r \leqslant n_0$, we define each term as follows:

$$X_{uv} = W(v)_1^r \times W(u)_{r+1}^{n_0},$$

$$Y_{uv} = \sum_{p=1}^{r-1} W(v, p) \times W(u)_{p+1}^{r},$$

$$Z_{uv} = \sum_{p=r+1}^{n_0-1} W(v, p) \times W(u)_{p+1}^{n_0}.$$

Algorithm `3-WOLF` consists mainly of two phases: A coarse-grained ordering phase followed by a fine-grained ordering phase. The initial coarse-grained phase partitions L_1 into disjoint sets and orders the partitions. This is done so that, given any pair $u, v \in L_1$ from two different partitions, if the partition of u is placed to the left of the partition of v, then $c_{uv} \leqslant 3c_{vu}$. Then the second phase orders the nodes within each partition independently, that is without considering the nodes in other partitions, so that given $u, v \in L_1$ from the same partition, if u is placed to the left of v, then $c_{uv} \leqslant 3c_{vu}$.

2.1. Phase-1: Coarse-grained ordering

To make the description easier for now we assume that G is a complete weighted bipartite graph, that is, some edge weights maybe zero. We later provide the details necessary to implement it more efficiently.

We partition L_1 into n_0 disjoint sets $\mathcal{P}_0, \ldots, \mathcal{P}_{n_0-1}$, where $L_1 = \bigcup_{0 \leqslant r \leqslant n_0-1} \mathcal{P}_r$. We define initial partition $\mathcal{P}_0$ as, $\mathcal{P}_0 = \{u \in L_1 \mid W(u)_1^0 \geqslant W(u)_2^{n_0}\}$. In general for $r \geqslant 1$, we define $\mathcal{P}_r$ as follows:

$$\mathcal{P}_r = \{u \in L_1 \mid W(u)_1^{r-1} < W(u)_{r+1}^{n_0} \text{ and } W(u)_1^r \geqslant W(u)_{r+2}^{n_0}\}.$$

Obviously each node in L_1 belongs to exactly one partition. The partitions are ordered in the increasing order of their indices from left to right. Algorithm 1 provides a pseudocode for Phase-1. The following lemma shows the correctness of the described partitioning and ordering:

Lemma 1. *Given $u \in \mathcal{P}_r$ and $v \in \mathcal{P}_q$, where $r < q$, after Phase-1 of Algorithm `3-WOLF` we have $c_{uv} \leqslant 3c_{vu}$.*

Proof. If $r = 0$ then $W(u)_2^{n_0} = 0$. Since $r < q$ we have $W(v)_2^{n_0} > 0$. This implies $c_{uv} \leqslant c_{vu}$ and the lemma holds trivially. Now assume $r > 0$. Since $u \in \mathcal{P}_r$ and $v \in \mathcal{P}_q$ by definition the following hold:

```
/*Initially each partition $\mathcal{P}_r$ where, $0 \leqslant r \leqslant n_0 - 1$, is empty.*/
for all $u \in L_1$ do
    $leftsum = 0$; $rightsum = W(u)_2^{n_0}$;
    for $r$ from 0 to $n_0 - 1$ do
        if $leftsum \geqslant rightsum$ then
            break;
        end if
        $leftsum = leftsum + W(u, r+1)$;
        $rightsum = rightsum - W(u, r+2)$;
    end for
    $\mathcal{P}_r = \mathcal{P}_r \cup \{u\}$;
end for
/*Partitions are ordered according to indices: $\mathcal{P}_0, \ldots, \mathcal{P}_{n_0-1}$*/
```

Algorithm 1. Coarse-grained ordering.

$$W(u)_1^{r-1} < W(u)_{r+1}^{n_0} \quad \text{and} \quad W(u)_1^r \geqslant W(u)_{r+2}^{n_0}, \tag{1}$$

$$W(v)_1^{q-1} < W(v)_{q+1}^{n_0} \quad \text{and} \quad W(v)_1^q \geqslant W(v)_{q+2}^{n_0}. \tag{2}$$

We show that $X_{uv}, Y_{uv}, Z_{uv} \leqslant X_{vu} + Y_{vu} + Z_{vu} = c_{vu}$. We have $W(u)_1^r \geqslant W(u)_{r+2}^{n_0}$ by (1). This implies $W(u)_1^{r+1} \geqslant W(u)_{r+1}^{n_0}$. Since $r < q$ by the first part of (2) we have $W(v)_1^r < W(v)_{q+1}^{n_0}$. Putting together we get

$$W(v)_1^r \times W(u)_{r+1}^{n_0} \leqslant W(u)_1^{r+1} \times W(v)_{q+1}^{n_0}.$$

Since $r < q$ the right side is at most $W(u)_1^r \times W(v)_{r+1}^{n_0} + W(u)_{r+1}^{r+1} \times W(v)_{q+1}^{n_0}$. The first term in this sum is equal to X_{vu}. The second term is at most Z_{vu}. Therefore $X_{uv} \leqslant c_{vu}$.

To prove it for Y_{uv}, we have

$$\sum_{p=1}^{r-1} W(v,p) \times W(u)_{p+1}^r \leqslant W(u)_2^r \times W(v)_1^{r-1}.$$

Since $r < q$ by (2) we have $W(v)_1^{r-1} < W(v)_{q+1}^{n_0}$ which further implies $W(v)_1^{r-1} < W(v)_{r+1}^{n_0}$. Putting together we have $Y_{uv} \leqslant X_{vu}$ and therefore $Y_{uv} \leqslant c_{vu}$.

Similarly for Z_{uv} we get

$$\sum_{p=r+1}^{n_0-1} W(v,p) \times W(u)_{p+1}^{n_0} \leqslant W(u)_{r+2}^{n_0} \times W(v)_{r+1}^{n_0-1}.$$

We have $W(u)_1^r \geqslant W(u)_{r+2}^{n_0}$ by (1) which implies $Z_{uv} \leqslant X_{vu}$ and therefore $Z_{uv} \leqslant c_{vu}$. $\square$

2.2. Phase-2: Fine-grained ordering

Let $\pi(\mathcal{P}_r)$ be a permutation of the nodes in $\mathcal{P}_r$. We define the following invariant:

Definition 2. Given a partition $\mathcal{P}_r$ and S such that $S \subseteq \mathcal{P}_r$, let $\pi(S)$ be a permutation of S. We call $\pi(S)$ a **partition invariant satisfying permutation (PISP)** if for any $u \in S$ and for all $v \in S \setminus \{u\}$ that are placed to the right of u in $\pi(S)$ the following inequalities hold:

$$W(v)_1^r \times W(u)_{r+1}^{n_0} \leqslant W(u)_1^r \times W(v)_{r+1}^{n_0}.$$

We show that an algorithm that orders $\mathcal{P}_r$ according to the partition invariant is appropriate for our purposes:

Lemma 3. *Let $\pi(\mathcal{P}_r)$ be a PISP of $\mathcal{P}_r$. For any pair $u, v \in \mathcal{P}_r$ where u is to the left of v in $\pi(\mathcal{P}_r)$ we have $c_{uv} \leqslant 3c_{vu}$.*

Proof. We show that $X_{uv}, Y_{uv}, Z_{uv} \leqslant X_{vu} \leqslant c_{vu}$. This is true for X_{uv} since $\pi(\mathcal{P}_r)$ is a PISP. To prove it for Y_{uv} we note that

$$\sum_{p=1}^{r-1} W(v,p) \times W(u)_{p+1}^r \leqslant W(u)_1^r \times W(v)_1^{r-1}.$$

From the definition of $\mathcal{P}_r$ we have $W(v)_1^{r-1} < W(v)_{r+1}^{n_0}$. Therefore,

$$\sum_{p=1}^{r-1} W(v,p) \times W(u)_{p+1}^r \leqslant W(u)_1^r \times W(v)_{r+1}^{n_0},$$

which implies $Y_{uv} \leqslant X_{vu}$. Finally for Z_{uv} we note that

$$\sum_{p=r+1}^{n_0-1} W(v,p) \times W(u)_{p+1}^{n_0} \leqslant W(u)_{r+2}^{n_0} \times W(v)_{r+1}^{n_0-1}.$$

From the definition of $\mathcal{P}_r$ we have $W(u)_1^r \geqslant W(u)_{r+2}^{n_0}$. Therefore,

$$\sum_{p=r+1}^{n_0-1} W(v,p) \times W(u)_{p+1}^{n_0} \leqslant W(u)_1^r \times W(v)_{r+1}^{n_0},$$

which further implies $Z_{uv} \leqslant X_{vu}$. $\square$

We show that we can construct a PISP efficiently. The following transitivity lemma will be helpful for further results.

```
/*Nodes in $\mathcal{P}_r$ have nonzero degrees. Initially $\pi(\mathcal{P}_r)$ is empty. */
if $|\mathcal{P}_r| = 1$ then
  $\pi(\mathcal{P}_r) = \{u\}$ where $u \in \mathcal{P}_r$
else
  Divide $\mathcal{P}_r$ into partitions of equal size $\mathcal{P}_{r1}, \mathcal{P}_{r2}$
  /*Solve for each partition.*/
  $\pi(\mathcal{P}_{r1}) =$ Fine-grained Ordering $\mathcal{P}_{r1}$
  $\pi(\mathcal{P}_{r2}) =$ Fine-grained Ordering $\mathcal{P}_{r2}$
  /*Merge the resulting permutations.*/
  New node $a$ s.t. $W(a)_1^r = -1, W(a)_{r+1}^{n_0} = 0$
  $\pi(\mathcal{P}_{r1}) = \pi(\mathcal{P}_{r1}) \circ \{a\}$
  $\pi(\mathcal{P}_{r2}) = \pi(\mathcal{P}_{r2}) \circ \{a\}$
  Let $u, v$ be the first nodes in $\pi(\mathcal{P}_{r1}), \pi(\mathcal{P}_{r2})$ respectively
  for $t \leftarrow 1$ to $|\mathcal{P}_r|$ do
    if $W(v)_1^r \times W(u)_{r+1}^{n_0} \leqslant W(u)_1^r \times W(v)_{r+1}^{n_0}$ then
      $\pi(\mathcal{P}_r) = \pi(\mathcal{P}_r) \circ \{u\}$
      $u = u'$ where $u'$ follows $u$ in $\pi(\mathcal{P}_{r1})$
    else
      $\pi(\mathcal{P}_r) = \pi(\mathcal{P}_r) \circ \{v\}$
      $v = v'$ where $v'$ follows $v$ in $\pi(\mathcal{P}_{r2})$
    end if
  end for
end if
```

Algorithm 2. Fine-grained ordering of $\mathcal{P}_r$.

Lemma 4. *Given $u, v, w \in \mathcal{P}_r$, each with degree at least one, assume the following inequalities hold for some j, $1 \leqslant j \leqslant n_0$:*

$$W(v)_1^j \times W(u)_{j+1}^{n_0} \leqslant W(u)_1^j \times W(v)_{j+1}^{n_0}, \tag{3}$$

$$W(w)_1^j \times W(v)_{j+1}^{n_0} \leqslant W(v)_1^j \times W(w)_{j+1}^{n_0}. \tag{4}$$

Then $W(w)_1^j \times W(u)_{j+1}^{n_0} \leqslant W(u)_1^j \times W(w)_{j+1}^{n_0}$.

Proof. Multiplying both sides of the inequality (3) in the statement of the lemma with $W(w)_1^j$ and replacing $W(v)_{j+1}^{n_0} \times W(w)_1^j$ with $W(v)_1^j \times W(w)_{j+1}^{n_0}$ we get:

$$W(v)_1^j \times W(u)_{j+1}^{n_0} \times W(w)_1^j \leqslant W(u)_1^j \times W(v)_1^j \times W(w)_{j+1}^{n_0}. \tag{5}$$

If $W(v)_1^j \neq 0$, we can divide both sides of (5) with $W(v)_1^j$ and the lemma holds. On the other hand if $W(v)_1^j = 0$, then $W(v)_{j+1}^{n_0} \neq 0$ since v has degree at least one. This implies $W(w)_1^j = 0$ and the lemma holds trivially. □

Lemma 4 together with the trivial antisymmetry and the totalness properties show that the operator described in Definition 2 induces a total order. Therefore any comparison sort on $\mathcal{P}_r$ produces a PISP and by Lemma 3 is appropriate for our 3-approximation algorithm. For sake of completeness we provide a pseudocode based on Mergesort in Algorithm 2 and a correctness proof below.

We assume all nodes in $\mathcal{P}_r$ have degree at least one, as nodes with degree zero can be placed arbitrarily. Assuming $\pi(\mathcal{P}_{r1}), \pi(\mathcal{P}_{r2})$ are PISPs, we need to prove the correctness of the merge procedure. We do so inductively. Let $\pi(\mathcal{P}_r)^t$ denote $\pi(\mathcal{P}_r)$ after t steps of the merge procedure. Let u', v' be the current nodes of $\pi(\mathcal{P}_{r1}), \pi(\mathcal{P}_{r2})$ respectively at the end of t steps. Let "$\circ$" denote the concatenation operator.

Lemma 5. *$\pi(\mathcal{P}_r)^t \circ \{u'\}$ and $\pi(\mathcal{P}_r)^t \circ \{v'\}$ are PISPs, where $0 \leqslant t \leqslant |\mathcal{P}_r|$.*

Proof. The proof is by induction on t. The base case of $t = 0$ holds trivially. Let u, v be the current nodes in $\pi(\mathcal{P}_{r1}), \pi(\mathcal{P}_{r2})$ respectively, at the beginning of step t. Assume inductively that $\pi(\mathcal{P}_r)^{t-1} \circ \{u\}$ and $\pi(\mathcal{P}_r)^{t-1} \circ \{v\}$ are PISPs.

Without loss of generality assume

$$W(v)_1^r \times W(u)_{r+1}^{n_0} \leqslant W(u)_1^r \times W(v)_{r+1}^{n_0}. \tag{6}$$

Note that this implies $v' = v$. We show the following:

1. $\pi(\mathcal{P}_r)^{t-1} \circ \{u\} \circ \{u'\}$ is a PISP: We note that

$$W(u')_1^r \times W(u)_{r+1}^{n_0} \leqslant W(u)_1^r \times W(u')_{r+1}^{n_0} \tag{7}$$

as u is to the left of u' in $\pi(\mathcal{P}_{r1})$ which is a PISP.

We need to show that $W(u')_1^r \times W(x)_{r+1}^{n_0} \leqslant W(x)_1^r \times W(u')_{r+1}^{n_0}$ for all $x \in \pi(\mathcal{P}_r)^{t-1} \circ \{u\}$. If $x = u$ the inequality is satisfied because of (7). On the other hand if $x \neq u$, we have $W(u)_1^r \times W(x)_{r+1}^{n_0} \leqslant W(x)_1^r \times W(u)_{r+1}^{n_0}$ since by the inductive hypothesis $\pi(\mathcal{P}_r)^{t-1} \circ \{u\}$ is a PISP. Using the transitivity property stated in Lemma 4, we combine this last inequality and that of (7) which implies $W(u')_1^r \times W(x)_{r+1}^{n_0} \leqslant W(x)_1^r \times W(u')_{r+1}^{n_0}$.

2. $\pi(\mathcal{P}_r)^{t-1} \circ \{u\} \circ \{v\}$ is a PISP: We need to show that $W(v)_1^r \times W(x)_{r+1}^{n_0} \leqslant W(x)_1^r \times W(v)_{r+1}^{n_0}$, for all $x \in \pi(\mathcal{P}_r)^{t-1} \circ \{u\}$. If $x = u$ the inequality is satisfied because of the initial assumption in (6). On the other hand if $x \neq u$ the inequality holds by the inductive hypothesis.

The lemma below is an immediate consequence of Lemma 5:

Lemma 6. *Let $\pi(\mathcal{P}_r)$ be the output permutation of Algorithm 2 applied on $\mathcal{P}_r$. $\pi(\mathcal{P}_r)$ is a PISP.*

The correctness of Phase-2 follows from the lemma above and Lemma 3:

Lemma 7. *Given $u, v \in \mathcal{P}_r$, where u is placed to the left of v after Phase-2 of Algorithm* `3-WOLF` *we have $c_{uv} \leqslant 3c_{vu}$.*

2.3. Tightness of the approximation ratio

We show that the approximation ratio analysis of Lemmas 1 and 7 are tight in the worst case.

Lemma 8. *There exist bipartite graph instances for which the bound of Lemma 1 is almost met.*

Proof. Fig. 1(a) shows the building block of such an instance. The lower layer shows L_0, whereas the upper layer is L_1. We have $u \in \mathcal{P}_q$, $v \in \mathcal{P}_r$ and u is placed to the left of v. The resulting number of crossings is $c_{vu} = w^2$, whereas $c_{uv} = 3w^2 - 2w\epsilon$. For sufficiently small ϵ this implies a ratio of almost 3 for c_{uv}/c_{vu}. The construction can easily be extended to $\Theta(n)$ nodes by introducing n copies of this block. Each node on L_1 is placed in a different partition as a result of Phase-1 of Algorithm `3-WOLF` on such an instance. □

Lemma 9. *There exist bipartite graph instances for which the bound of Lemma 7 is almost met.*

Proof. Fig. 1(b) shows the building block of such an instance. We have $u, v \in \mathcal{P}r$. We pick ϵ_v slightly smaller than ϵ. The construction can be extended to $\Theta(n)$ nodes by introducing n copies of node v. Phase-2 of Algorithm `3-WOLF` places u to the left of each copy of v as $W(v)_1^r \times W(u)_{r+1}^{n_0} < W(u)_1^r \times W(v)_{r+1}^{n_0}$. For sufficiently small ϵ, this implies $3nw^3 + \Theta(nw^2)$ crossings as a result of Phase-2 of Algorithm `3-WOLF`, whereas we get $nw^3 + \Theta(nw^2)$ crossings in the optimum placement, where u is placed to the right of all copies of v. □

2.4. Running time of Algorithm `3-WOLF`

In order to implement Phase-1 efficiently we note that a node $u \in L_1$ may not be connected to all the nodes in L_0. Let N_u denote the set of neighbors of u. We assume the nodes in L_0 are labeled from 1 to $|L_0|$ and that nodes in N_u are already in sorted order according to these labels. We compute the sum $\sum_{i=1}^{|N_u|} W(u, N_u[i])$ exactly once at the beginning. Here $N_u[i]$ indicates the ith neighbor's label. Going through every neighbor of u we increment *leftsum* value each time, and decrement the *rightsum* value only whenever necessary, where leftsum indicates $W(u)_1^k$ and rightsum indicates $W(u)_{k+2}^{n_0}$, for $0 \leqslant k < |N_u|$. The partition of u is found once the leftsum value is greater than or equal to the rightsum value. The

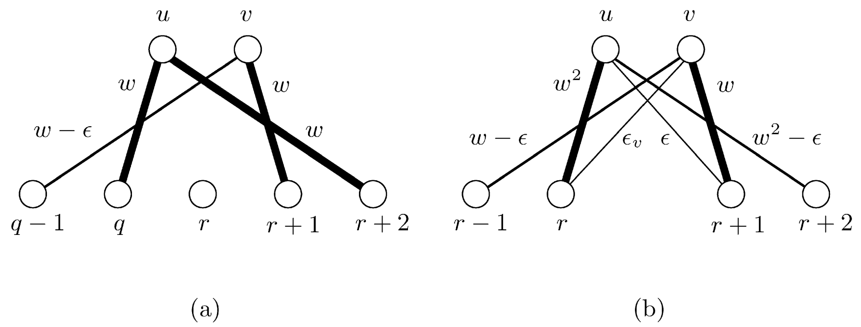


Fig. 1. (a) Building block of the worst case instance for Phase-1; (b) Building block of the worst case instance for Phase-2.

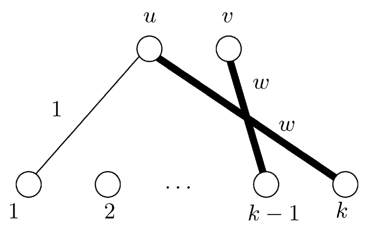


Fig. 2. Building block of the worst case instance for W-BARY.

running time required by this implementation of Phase-1 is $O(|E| + |L_0| + |L_1|)$. We assume the values $W(u)_1^r, W(u)_{r+1}^{n_0}$ are also recorded during Phase-1, once they are computed. Therefore following the description of Phase-2 in Algorithm 2, the second phase requires time $O(|L_0| + |L_1| \log |L_1|)$. The theorem below then follows this discussion and the correctness lemmas presented in Lemmas 1 and 7:

Theorem 10. *Given a bipartite graph $G = (L_0, L_1, E)$, Algorithm* 3-WOLF *3-approximates WOLF in time $O(|E| + |L_0| + |L_1| \log |L_1|)$.*

3. Weighted modifications of common methods for OLF

In addition to 3-WOLF, we devise modifications of four well-known methods suggested previously for OLF: The barycenter, median, GRE heuristics, and the penalty graph method.

3.1. W-BARY: Weighted barycenter

The original barycenter method assigns the x-coordinate of each $u \in L_1$ as the average of the x-coordinates of its neighbors [21]. In W-BARY edge weights are introduced to this average. That is, the x-coordinate of u is set to

$$\frac{\sum_{i=1}^{|N_u|}(W(u, N_u[i]) \times N_u[i])}{\sum_{i=1}^{|N_u|} W(u, N_u[i])}.$$

The approximation ratio of the barycenter method applied in OLF settings is $\Theta(\sqrt{|L_0|})$ [7]. In contrast to this, we show that there exist instances of WOLF for which W-BARY produces outputs where the approximation ratio is $\Omega(|L_0| + |L_1|)$. Interestingly such instances are plausible even when degrees are restricted to 2. We note that in OLF settings the barycenter method achieves an optimal solution under this restriction [17].

Lemma 11. *There exist bipartite graph instances for which* W-BARY *has an approximation ratio of $\Omega(|L_0| + |L_1|)$.*

Proof. Fig. 2 shows the building block of the construction of an instance where the performance ratio of W-BARY is $\Omega(|L_0| + |L_1|)$. We pick $w = k - 2 - \epsilon$. For sufficiently small ϵ, algorithm W-BARY places u to the left of v which introduces w^2 crossings, whereas the optimum placement would lead to only w crossings. The construction can be generalized to $\Theta(n)$ nodes in both L_0, L_1 by introducing $\Theta(n)$ copies of the pair u, v in L_1 and $\Theta(n)$ new nodes in L_0. □

3.2. W-MEDBARY: Weighted median + barycenter

The original median algorithm in OLF settings is described by Eades and Wormald [7]. It assigns x-coordinate of $u \in L_1$ to be the median of the x-coordinates of the nodes in N_u. If two nodes are assigned the same median, then one of them is placed to the left randomly, except when one has odd degree, and the other even, in which case the odd degree node is placed to the left. The weighted version we propose, W-MEDBARY, in essence is similar to 3-WOLF. Algorithm W-MEDBARY also proceeds in two phases. In the coarse-grained phase of W-MEDBARY we first decide on the partition of each node $u \in L_1$. Node u is placed in $\mathcal{P}_r$, where r is the smallest integer value such that $W(u)_1^r \geqslant W(u)_{r+1}^{n_0}$. The partitions are then ordered from left to right in the increasing order of their indices. In the second phase of W-MEDBARY we apply W-BARY on each partition $\mathcal{P}_r$.

We note that Phase-1 of W-MEDBARY is analogous to the median assignment in the OLF settings, assuming the medians are different. Consider the unweighted graph constructed by replacing each neighbor $\mathcal{N}_u[i]$ of $u \in L_1$ with $\mathcal{W}(u, \mathcal{N}_u[i])$ artificial nodes and connecting each one to u with an unweighted edge. Let x be the artificial node that is computed as the median of u as a result of applying the original median algorithm on the unweighted graph. Then the median node (the partition $\mathcal{P}_r$) picked by the first phase of W-MEDBARY is the one that is replaced by the artificial nodes including x. Since the original median algorithm provides a guaranteed constant approximation ratio of 3, it is natural to expect a good performance from the first phase of W-MEDBARY. In fact we can prove that nodes in different partitions are placed appropriately, up to the constant approximation ratio of 3, after this initial phase.

Lemma 12. *Given $u \in \mathcal{P}_q$ and $v \in \mathcal{P}_r$, where $q < r$, after Phase-1 of Algorithm* W-MEDBARY *we have $c_{uv} \leqslant 3c_{vu}$.*

Proof. Let $A = W(u)_1^{q-1}$, $X = W(u,q)$, $B = W(u)_{q+1}^{n_0}$. Similarly let $C = W(v)_1^{r-1}$, $Y = W(v,r)$, $D = W(v)_{r+1}^{n_0}$. It follows that,

$$c_{uv} \leqslant AC + CB + DB + CX + BY,$$
$$c_{vu} \geqslant AD + AY + DX + XY.$$

We observe that $B \leqslant A + X$ and $C \leqslant D + Y$ after Phase-1 of W-MEDBARY. Replacing B with $A + X$ and C with $D + Y$ in the inequality for c_{uv}, we get $c_{uv} \leqslant 3AD + 3AY + 3DX + 3XY$, which implies $c_{uv} \leqslant 3c_{vu}$. □

For the second phase, in OLF settings, the median algorithm simply checks the node degrees to order the nodes within each partition to guarantee the same approximation ratio. However similar reasoning does not seem to apply to WOLF.

3.3. W-GRE: Weighted GRE

The GRE algorithm described for OLF greedily assigns $u \in L_1$ as the next node to place in the rightmost position [23]. It does so by choosing node u that minimizes

$$\frac{\sum_{v \in L_1'} c_{uv}}{\sum_{v \in L_1'} \min(c_{uv}, c_{vu})}$$

among all not yet placed nodes, denoted by L_1', where $L_1' = L_1$ initially. The W-GRE algorithm works the same way, except edge weights are taken into account while computing c_{uv}.

3.4. W-PM: Weighted penalty minimization

In OLF settings the PM algorithm starts by constructing a weighted directed graph called the *penalty graph*. The node set is that of L_1. An edge with weight $c_{vu} - c_{uv}$ from u to v is inserted in the penalty graph if $c_{uv} < c_{vu}$. The algorithm then seeks for a minimum *feedback arc set* (FAS) in the penalty graph [21]. A feedback arc set in a directed graph is a subset of edges which contains at least one edge from each directed cycle of the graph. In the weighted version of the problem the goal is to find a feedback arc set with minimum total weight. Demetrescu and Finocchi propose an implementation of the PM method based on their algorithm for approximating FAS in weighted directed graphs [5]. The W-PM method is based on their implementation, except as with W-GRE, the computation of c_{uv} takes into account the edge weights.

Remark 13. All algorithms assume N_u is sorted for $u \in L_1$. Both W-BARY and W-MEDBARY run in linear time. The running time of W-GRE is $O(|E|^2 + |L_1|^2)$ and that of W-PM is $O(|E|^2 + |L_1|^4)$. Both W-GRE and W-PM require the computation of a cross table. All c_{uv} values are retrieved from this table which is computed beforehand. A straightforward implementation of this computation requires time $O(|E|^2)$. In OLF settings it can be implemented in time $O(|E| \times |L_1|)$. This improvement is based on the observation that each crossing increments the total crossings by the same amount of 1. However this is no longer true in WOLF settings. Therefore our construction of the cross table requires $O(|E^2|)$ time. We note that this discrepancy does not lead to any noticeable affect on the actual CPU times required by the suggested algorithms, W-GRE and W-PM.

4. Experimental results and discussion

We implemented all the algorithms in C++ using the LEDA library [16]. For experimental setup we made use of GSL [10] and Stanford Graphbase [14] libraries. The implementations are freely available in [1]. The experiments are performed on a P4 3.2 GHz of CPU with 1 GB of RAM, 1 MB of layer 2 cache and 16 KB of layer 1 cache.

4.1. Input graph generation and experimental parameters

Input graph generation consists mainly of two phases. The first phase creates an unweighted graph. We use three different methods to do this. First one is random unweighted graph generation. This is done using the random bipartite graph generator from GraphBase [14]. We note that the graphs generated are not necessarily connected. In OLF settings, depending on whether the input graph is connected, a given heuristic may perform quite differently [20]. To see whether the same holds in WOLF settings, we apply a second method that definitely generates a connected random unweighted graph. Given the nodes on the two layers, initially a minimum spanning tree based on random costs is generated and extra edges are inserted randomly if necessary [20]. Previous experimental studies on OLF consider not only random bipartite graphs but also some specific input graph classes [6,13]. Similarly, our third method generates unweighted graphs from a specific class, Warfield graphs [22].

The second phase of the graph generation assigns random positive weights to the edges according to a specific probability distribution. We use two distributions for random weights: the uniform distribution and the t-distribution. The former is a popular choice in random number generation. We use the latter as it is capable of creating both the normal (Gaussian) distribution where the outliers are downweighted, and the distributions where the heavy tailed data is also included. We use two versions of the t-distribution. According to the shape of their probability distribution functions we call them symmetric and asymmetric versions. The t-distribution is symmetric in nature but when we cut the negative tail of it, in order to generate positive weights, it becomes an asymmetric distribution. In the symmetric version we shift the t-distribution to right in x-axis to provide positive weights. In both versions the t-distribution takes a single parameter p.

We note that since we generate the random graphs using the Stanford GraphBase Library and store the seeds of the generated graphs, this phase of the input graph generation is reproducible. For the random weight generation phase, we use GSL to generate random numbers for a given distribution and seed values, and we store the seeds of weight generation as well. We generated the seed sequence for GraphBase from a generator seed "WOLF" that is 9653. So the whole process of random weighted bipartite graph generation is reproducible.

For the case where random input graphs (connected or not) are generated in the first phase, we have two different settings for the graph sizes. The first one is 100×100, 100 nodes on the fixed layer and 100 nodes on the free layer, while the second setting is 30. In addition to the node count on the layers *edge density* is another parameter used in this case. It is defined as the ratio of the number of edges in the generated graph to the number of edges in a complete bipartite graph with the same number of nodes in both layers. In our experiments with the randomly generated graphs, we use edge density values varying between 0.001 and 0.5.

4.2. Experiments and discussion of results

4.2.1. Performance measures

We have two performance measures. For a given node count and edge density we compute *crossing*/*LB* for each algorithm to measure the quality of its solution to WOLF. Here *crossing* indicates the total weighted crossings that arise in the output layout of an algorithm. For a given input graph a trivial lower bound on the number of crossings is $LB = \sum_{u,v \in L_1} \min(c_{uv}, c_{vu})$. For each instance (determined by the number of nodes on each layer and the edge density) the experiment is repeated 50 times on different random graphs and an average *crossing*/*LB* value is computed. The second performance measure is the running time required by each algorithm.

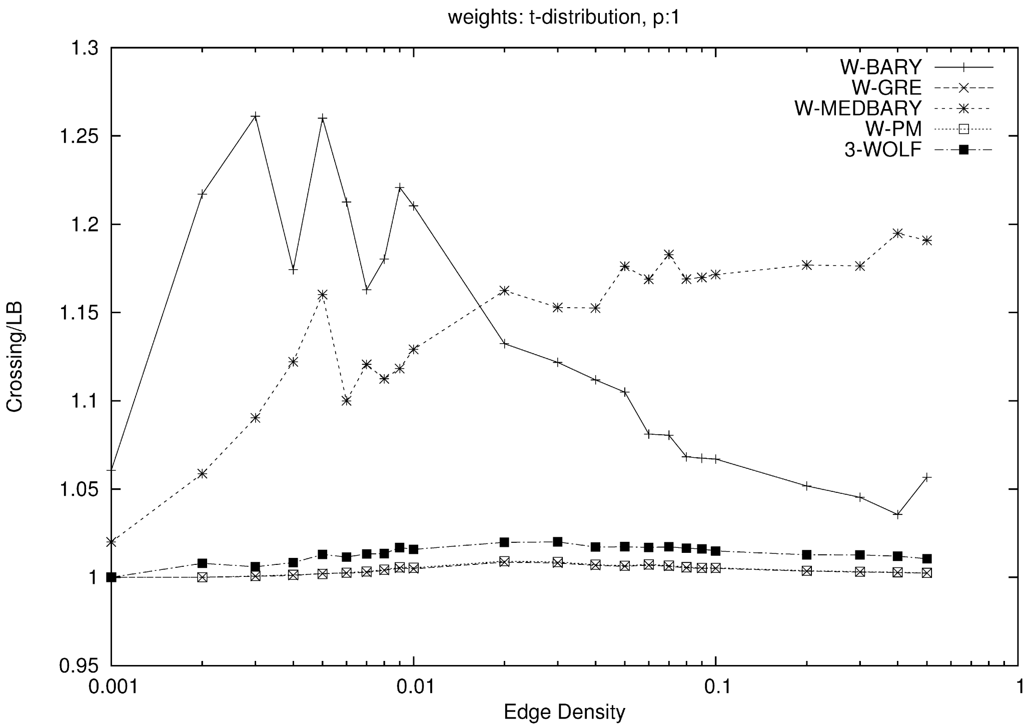


Fig. 3. *Crossing*/*LB* values for random graph instances of size 100×100. Weight assignment is via the t-distribution with $p = 1$.

Table 1
Detailed results including minimum, average, maximum *crossing*/*LB* values for random graph instances of size 100 × 100. Weight assignment is via the t-distribution with $p = 1$

ED	W-PM	W-GRE	3-WOLF	W-BARY	W-MEDBARY
	1.000000	1.000000	1.000000	1.000000	1.000000
0.001	1.000000	1.000000	1.000000	1.060584	1.020000
	1.000000	1.000000	1.000000	2.500000	2.000000
	1.000000	1.000000	1.000000	1.016149	1.000000
0.005	1.001922	1.002208	1.012902	1.260082	1.160157
	1.016667	1.027778	1.056676	2.905852	1.862869
	1.000000	1.000000	1.002254	1.055229	1.006374
0.010	1.005381	1.004866	1.015827	1.210420	1.129096
	1.018227	1.021520	1.057720	2.048758	1.290548
	1.001226	1.001138	1.002443	1.037737	1.085880
0.050	1.006573	1.006247	1.017305	1.104880	1.176186
	1.015474	1.014919	1.035880	1.531312	1.402872
	1.001244	1.001254	1.002948	1.026231	1.098792
0.100	1.005328	1.005057	1.014907	1.067029	1.171494
	1.012627	1.012838	1.051906	1.195283	1.358306
	1.001409	1.001344	1.003227	1.015300	1.097249
0.300	1.003200	1.003003	1.012692	1.045358	1.176336
	1.005588	1.005157	1.033871	1.232040	1.281945
	1.000205	1.000213	1.000897	1.008193	1.075158
0.500	1.002531	1.002391	1.010511	1.056668	1.190872
	1.006463	1.005917	1.027195	1.329184	1.492033

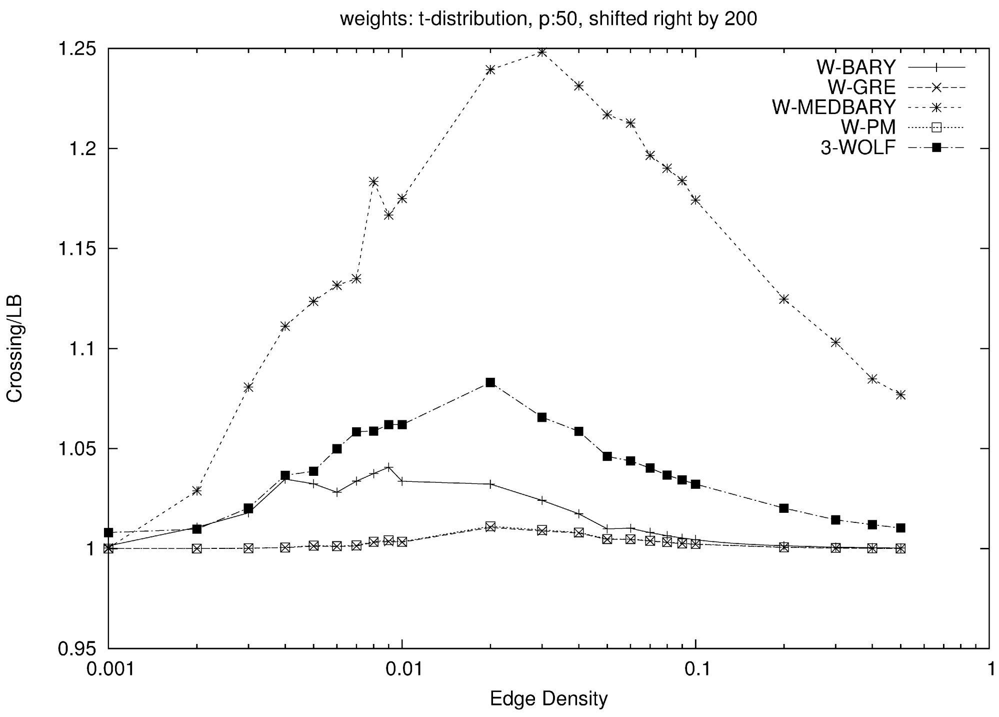
4.2.2. Experimental setup and results

Fig. 3 demonstrates the results from our first set of experiments on 100 × 100 graphs with edge densities varying from 0.001 to 0.5. The unweighted graphs are generated randomly using the GraphBase instances. The weights are assigned randomly using the t-distribution with $p = 1$ (asymmetric version). Note that the x-axis of the figure is plotted in log-scale. In this setup 3-WOLF outperforms both W-BARY and W-MEDBARY heuristics. Additionally, the output quality of 3-WOLF is almost the same as that of W-PM and W-GRE which produce total weighted crossings nearly equal to the lower bound value. W-MEDBARY performs better than W-BARY for sparse graph instances, whereas the roles are swapped as the graphs become denser. Detailed results of this setup are presented in Table 1. The minimum, average, and the maximum *crossing*/*LB* values for a given instance are represented respectively at each row's top, middle, and bottom entries. The leftmost column *ED* in the table represents the edge density of an instance class.

We note that when the experiments are repeated under the same settings but with graphs of size 30 × 30 the results are almost the same. The only difference is that for very sparse graphs (edge density close to 0.001) all algorithms have almost identical performances, since the number of edges is close to 0. For the second set of experiments unweighted graphs are again generated randomly using the GraphBase instances. The size is again 100 × 100. The random weight assignment uses the t-distribution but this time the distribution is shifted to the right along the x-axis by an appropriate value m and the distribution parameter p is set to 50. This simulates a normal distribution with mean m. W-PM and W-GRE are again the best heuristics for all edge density values, see the plot at the top of Fig. 4. In this setup, for sparse graphs where the edge density is between 0.001 and 0.01, 3-WOLF and W-BARY have similar output quality in terms of total weighted crossings in the output layouts. However W-BARY performs better than 3-WOLF for denser graphs. Its performance is almost the same as that of W-PM and W-GRE starting with edge density 0.01. In all edge density values W-MEDBARY has the worst performance. Implementing the random weight assignment phase with the uniform weight distribution instead of the normal distribution, the same set of experiments are repeated. We note that the results are almost the same as the normal distribution case. We conjecture that similar performance results should be obtained for all random weight assignments that use symmetric probability distributions as such.

It is interesting to see how the heuristics designed for WOLF perform in OLF settings. The bottom plot of Fig. 4 shows the performance results of the algorithms under unit weighting. Note the similarity between the two plots in the figure. The results obtained under unit weighting are similar to the cases where the weight distribution is via some symmetric probability distribution (normal, uniform). We believe this is further evidence of the stated conjecture.

In the previous experimental setups the input graphs are not necessarily connected. In the OLF settings the performances of heuristics vary depending on whether the input graph is connected or not [20]. The next set of experiments are conducted to see if the same holds in WOLF settings. Fig. 5 shows the performance results of the algorithms under discussion with random connected graphs. The first phase of the graph generation creates the random unweighted graph using the implementation of [20]. The weights are then distributed using the t-distribution with $p = 1$.



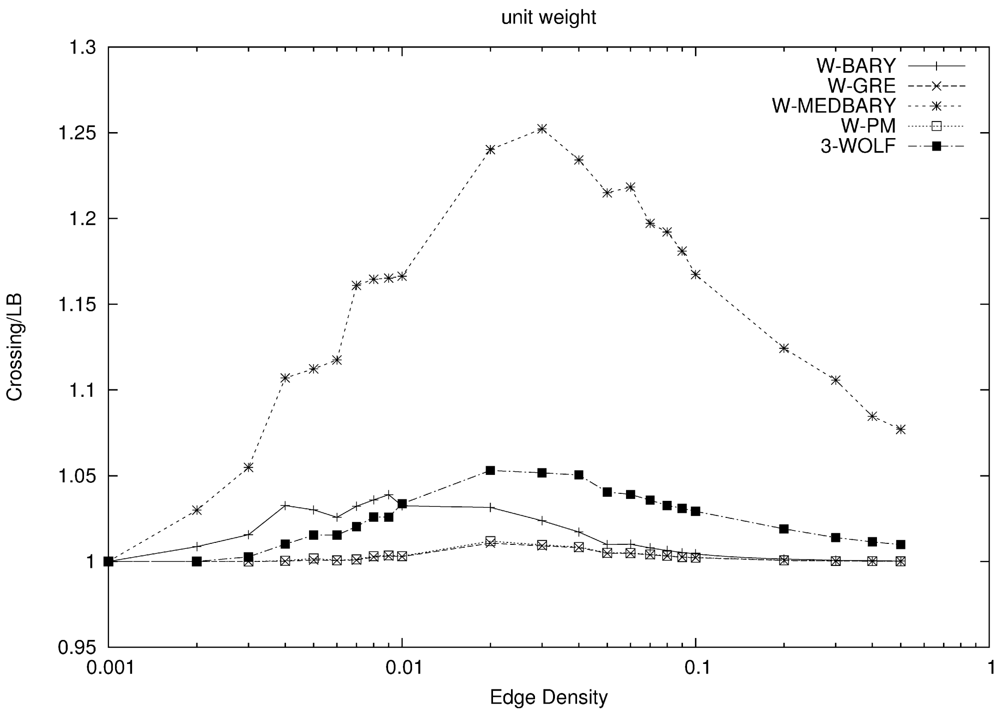


Fig. 4. Top: *Crossing*/*LB* values for random graph instances of size 100 × 100. Weight assignment is via the t-distribution with $p = 1$, shifted by $m = 200$ (normal distribution). Bottom: *Crossing*/*LB* values for random graph instances of size 100 × 100. The graphs are unweighted.

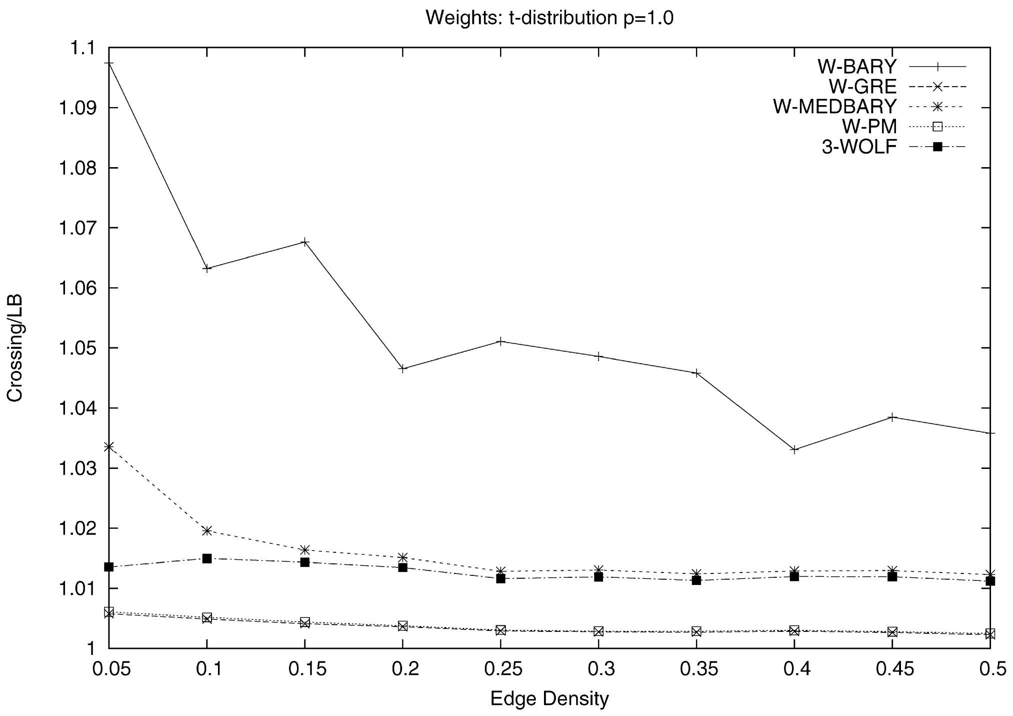


Fig. 5. *Crossing/LB* values for *connected* graph instances of size 100 × 100. Weight assignment is via the t-distribution with $p = 1$.

4.2.3. Discussion of results

The results are similar to those obtained under the same settings, except the random unweighted graphs are not required to be connected, see Fig. 3. The only difference is that W-MEDBARY's performance does not degrade with increasing density and is closer to the performance of 3-WOLF. We note that the edge density values for this setup starts from 0.05 to guarantee connectivity. Repeating the same setup of experiments with the use of normal distribution in the random weight assignment phase provides results similar to the ones presented in Fig. 4. This indicates that in WOLF settings the performances of the heuristics do not depend noticeably on whether the input graphs are connected, but rather on the type of the distribution chosen for the random weight assignment phase.

Each of the previous setups begins with a randomly generated unweighted graph. For a fixed graph size and edge density, 50 instances are generated randomly and the discussed results reflect the average of the results obtained from these instances. It is important to evaluate the heuristics for structured classes of graphs as well as randomly generated ones. A common class of graphs covered in previous OLF studies is the Warfield instances. The size of each instance is defined as $n_0 = k$, $n_1 = 2^k - 1$ for some positive integer k. There exists an edge between $i \in L_0$ and $j \in L_1$ if the ith bit of j represented in binary is 1.

We present the results of the WOLF heuristics applied to these instances and compare them with the results obtained from randomly generated instances of the same size and density in Table 2. Each row in the table has four values for each algorithm. The first value represents the *crossing/LB* values when Warfield instances are assigned random weights with t-distribution ($p = 1$). The second is the same value under the same settings except the t-distribution parameters are changed ($p = 50$ and shifted by 200) to provide normally distributed weights.

The third represents the same value for randomly generated instances using GraphBase instances of the same size and density under weight assignment using t-distribution ($p = 1$). The final value is obtained under the same setting as the third except the weights are generated according to normal distribution. For a given heuristic there is no noticeable difference between the results obtained for settings using the Warfield instances or the randomly generated instances.

Our final experiments measure the performance of the heuristics in terms of the time required to compute the resulting layouts, see Fig. 6. We note that y-axis is log-scale. As expected W-PM and W-GRE have the worst performances in terms of the required CPU times, making them infeasible to employ for very large input graphs. 3-WOLF have running times close to W-BARY and W-MEDBARY. 3-WOLF has running times comparable to those of W-BARY and W-MEDBARY. The slight differences do not appear to produce any noticeable effects.

5. Open problems

A similar version of the unweighted, one layer free bipartite crossing minimization problem, OLF, is one where both layers are freely permutable. A common technique employed in this case is to iteratively apply a solution to OLF, while

Table 2
Warfield instances versus random graphs of the same size and edge density

k	W-PM	W-GRE	3-WOLF	W-BARY	W-MEDBARY
	1.000000	1.000000	1.005137	1.006861	1.021493
3.000	1.000000	1.000000	1.007938	1.000000	1.063202
	1.000930	1.000930	1.018881	1.013235	1.117774
	1.000000	1.000000	1.026150	1.003021	1.159364
	1.000000	1.000000	1.007917	1.013190	1.014833
4.000	1.000000	1.000000	1.048925	1.000000	1.069603
	1.000137	1.000137	1.008947	1.020839	1.190544
	1.000000	1.000000	1.023468	1.002216	1.153913
	1.000260	1.000260	1.018500	1.020945	1.026037
5.000	1.000000	1.000000	1.030098	1.000000	1.051887
	1.000000	1.000000	1.014365	1.051943	1.127604
	1.000001	1.000001	1.014966	1.000904	1.127942
	1.000083	1.000083	1.022281	1.013835	1.023828
6.000	1.000000	1.000000	1.036168	1.000000	1.051533
	1.000098	1.000093	1.013213	1.008467	1.231747
	1.000007	1.000007	1.011626	1.000472	1.085030
	1.000089	1.000089	1.019979	1.019995	1.019910
7.000	1.000000	1.000000	1.039826	1.000000	1.047395
	1.000164	1.000163	1.012409	1.018831	1.169043
	1.000001	1.000001	1.011364	1.000131	1.065723
	1.000259	1.000259	1.024261	1.021284	1.023823
8.000	1.000000	1.000000	1.046756	1.000000	1.049881
	1.000125	1.000125	1.014637	1.004734	1.157756
	1.000001	1.000001	1.006926	1.000090	1.043716
	1.000371	1.000378	1.025076	1.021468	1.025241
9.000	1.000000	1.000000	1.041861	1.000000	1.049563
	1.000174	1.000168	1.013721	1.017097	1.155955
	1.000004	1.000004	1.004661	1.000032	1.033928

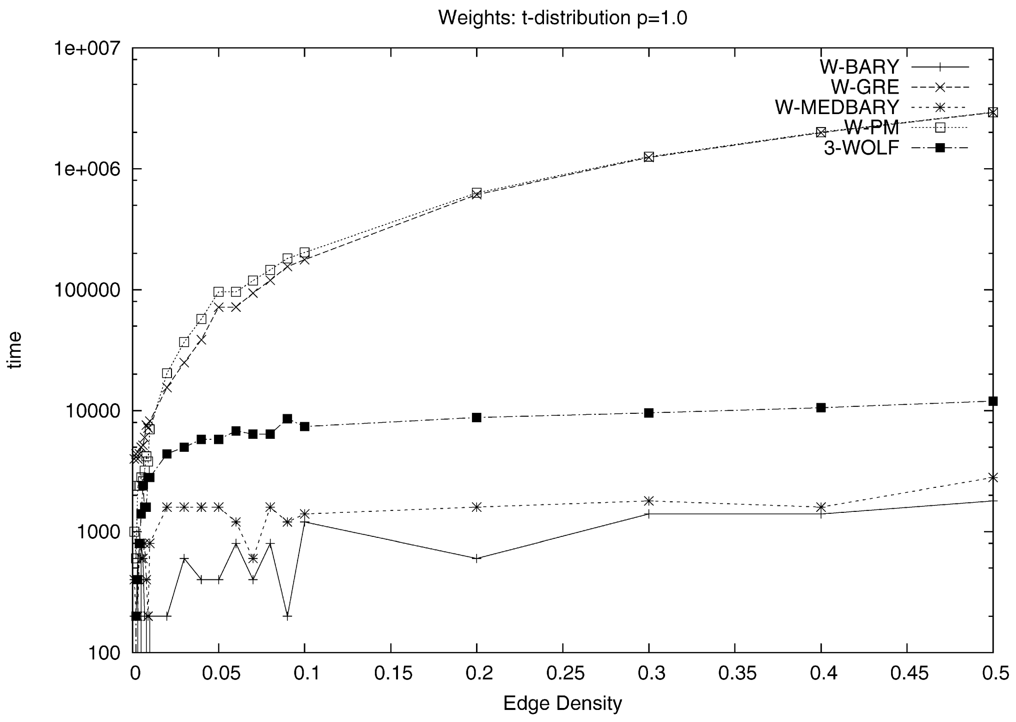


Fig. 6. Runtime plots of heuristics under evaluation. Graph instances of size 100 × 100. Weight assignment is via the t-distribution with $p = 1$.

alternating the fixed layer at each iteration. Analogous problem definitions and solution techniques could be applied to the weighted bipartite graphs as well. Future work in this direction would include applying WOLF heuristics presented in this paper within this iterative solution technique and conducting experiments to see whether similar results are achieved.

Another problem related to OLF is that of minimizing edge crossings in multi-layered drawings of graphs. A layer-by-layer sweeping technique is common for this problem. Fixing the topmost layer the next layer is permuted using a suitable OLF heuristic. Then the permuted layer is fixed at the resulting permutation and the following layer is permuted the same way. This is iterated until all the layers are swept from top to bottom. The sweeping is repeated going from bottom layers to the top, and back and forth if necessary. Considering the multi-layered version of the problem in weighted settings is another important open research problem. Implementing the presented WOLF heuristics, each employed separately within this iterative sweeps technique, and evaluating their performances for the multi-layered problem is left as future work.

References

[1] http://www2.isikun.edu.tr/personel/cesimerten/wolf.rar.

[2] T. Biedl, F.J. Brandenburg, X. Deng, Crossings and permutations, in: Proceedings of Graph Drawing (GD '05), in: LNCS, Springer, 2006, pp. 1–12.

[3] U. Brandes, T. Dwyer, F. Schreiber, Visualizing related metabolic pathways in two and a half dimensions (long paper), in: Proceedings of Graph Drawing (GD '03), in: LNCS, Springer, 2004, pp. 111–122.

[4] U. Brandes, D. Wagner, Visone—analysis and visualization of social networks, in: Graph Drawing Software, Springer, 2003, pp. 321–340.

[5] C. Demetrescu, I. Finocchi, Breaking cycles for minimizing crossings, J. Exp. Algorithmics 6 (2001) 2.

[6] S. Dresbach, A new heuristic layout algorithm for dags, in: Operations Research Proceedings, Springer, 1994, pp. 121–126.

[7] P. Eades, N.C. Wormald, Edge crossings in drawings of bipartite graphs, Algorithmica 11 (1994) 379–403.

[8] I. Finocchi, Layered drawings of graphs with crossing constraints, in: COCOON '01: Proceedings of the 7th Annual International Conference on Computing and Combinatorics, London, UK, Springer-Verlag, 2001, pp. 357–367.

[9] M. Forster, Applying crossing reduction strategies to layered compound graphs, in: Proceedings of Graph Drawing (GD '02), in: LNCS, Springer, 2002, pp. 276–284.

[10] M. Galassi, GNU Scientific Library Reference Manual, revised second ed.

[11] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1979.

[12] M.R. Garey, D.S. Johnson, Crossing number is NP-complete, SIAM J. Algebraic Discrete Methods 4 (3) (1983) 312–316.

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

[14] D.E. Knuth, The Stanford Graphbase, Addison Wesley, 1993.

[15] X.Y. Li, M.F. Stallmann, New bounds on the barycenter heuristic for bipartite graph drawing, Inform. Process. Lett. 82 (6) (2002) 293–298.

[16] K. Mehlhorn, S. Näher, The LEDA Platform of Combinatorial and Geometric Computing, Cambridge University Press, 1999.

[17] X. Munoz, W. Unger, I. Vrto, One sided crossing minimization is np-hard for sparse graphs, in: GD '01: Revised Papers from the 9th International Symposium on Graph Drawing, London, UK, Springer-Verlag, 2002, pp. 115–123.

[18] H. Nagamochi, An improved bound on the one-sided minimum crossing number in two-layered drawings, Discrete Comput. Geom. 33 (4) (2005) 569–591.

[19] B.S. Smith, S.K. Lim, Qca channel routing with wire crossing minimization, in: GLSVSLI '05: Proceedings of the 15th ACM Great Lakes Symposium on VLSI, ACM Press, 2005, pp. 217–220.

[20] M. Stallmann, F. Brglez, D. Ghosh, Heuristics, experimental subjects, and treatment evaluation in bigraph crossing minimization, J. Exp. Algorithmics 6 (2001) 8.

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

[22] J.N. Warfield, Crossing theory and hierarchy mapping, IEEE Trans. Syst. Man. Cybernet. 7 (7) (1977) 502–523.

[23] A. Yamaguchi, A. Sugimoto, An approximation algorithm for the two-layered graph drawing problem, in: Proceedings of 5th Annual Intl. Conf. on Computing and Combinatorics (COCOON '99), in: LNCS, Springer, 1999, pp. 81–91.