Experiments on the Minimum Linear Arrangement Problem

Jordi Petit
Universitat Politècnica de Catalunya

This paper deals with the Minimum Linear Arrangement problem from an experimental point of view. Using a testsuite of sparse graphs, we experimentally compare several algorithms to obtain upper and lower bounds for this problem. The algorithms considered include Successive Augmentation heuristics, Local Search heuristics and Spectral Sequencing. The testsuite is based on two random models and "real life" graphs. As a consequence of this study, two main conclusions can be drawn: On one hand, the best approximations are usually obtained using Simulated Annealing, which involves a large amount of computation time. Solutions found with Spectral Sequencing are close to the ones found with Simulated Annealing and can be obtained in significantly less time. On the other hand, we notice that there exists a big gap between the best obtained upper bounds and the best obtained lower bounds. These two facts together show that, in practice, finding lower and upper bounds for the Minimum Linear Arrangement problem is hard.

1. INTRODUCTION

A *layout* or a *linear arrangement* of an undirected graph $G = (V, E)$ with $|V| = n$ is a one-to-one function $\varphi : V \to \{1, \ldots, n\}$. The *Minimum Linear Arrangement* problem (MinLA) is a combinatorial optimization problem formulated as follows: Given a graph $G = (V, E)$, find a layout φ that minimizes

$$\text{LA}(G, \varphi) = \sum_{uv \in E} |\varphi(u) - \varphi(v)|.$$

Figure 1 shows a minimal linear arrangement for an square mesh. From now on, we consider that the set of nodes of an input graph G is $V(G) = \{1, \ldots, n\}$ and that $|E(G)| = m$. As a consequence, a layout φ can also be seen as a permutation. Without lost of generality, we will assume that the input graph is connected and without self-loops.

MinLA is an interesting and appealing problem that belongs to the family of layout problems; see [Díaz et al. 2002] for a survey. It appears as problem number GT42 in [Garey and Johnson 1979] and as GT40 in [Ausiello et al. 1999]. The Minimum Linear Arrangement problem has received some alternative names in the literature, such as the *optimal linear ordering*, the *edge sum problem* or the *minimum-1-sum*.

This problem was originally motivated as an abstract model in the design of VLSI layouts. Given a set of modules, the VLSI layout problem consists in placing the modules on a board in a non-overlapping manner and wiring together the terminals

This research was partially supported by the IST Programme of the EU under contract numbers IST-1999-14186 (ALCOM-FT) and IST-2001-33116 (FLAGS), and by the CICYT project TIC2002-04498-C05-03 (TRACER).

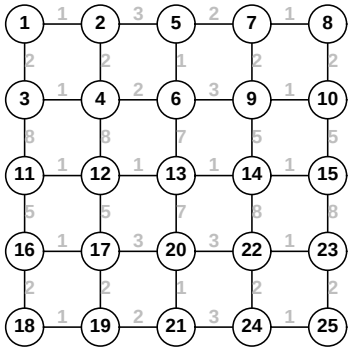


Fig. 1. An optimal linear arrangement for the 5×5 mesh. The black numbers in the nodes show the labeling. The grey numbers represent the incurred costs for each edge, whose sum is 117.

on the different modules according to a given wiring specification and in such a way that the wires do not interfere among them. There are two stages in VLSI layout: placement and routing. The placement problem consists in placing the modules on a board; the routing problem consists in wiring together the terminals on different modules that should be connected. Several approaches to solve the placement phase use the Minimum Linear Arrangement problem in order to minimize the total wire length [Harper 1970; Adolphson and Hu 1973].

MinLA is also related to graph drawing: A bipartite drawing (or 2-layer drawing) is a graph representation where the nodes of a bipartite graph are placed in two parallel lines and the edges are drawn with straight lines between them. The bipartite crossing number of a bipartite graph is the minimal number of edge crossings over all bipartite drawings. [Pach et al. 1996] show that for a large class of bipartite graphs, reducing the bipartite crossing number is equivalent to reducing the total edge length, that is, to the Minimum Linear Arrangement problem. Moreover, an approximate solution of MinLA can be used to generate an approximate solution to the Bipartite Crossing Number problem.

The MinLA problem has also received attention as an over-simplified model of some nervous activity in the cortex [Mitchison and Durbin 1986] and has also been shown to be relevant to single machine job scheduling [Adolphson 1977; Ravi et al. 1991].

The MinLA problem is known to be **NP**-hard and its decision version is **NP**-complete [Garey et al. 1976]. However, there exist polynomial time algorithms to compute exact solutions for some particular classes of graphs, which we survey in Table 1. Notwithstanding these positive results, the decision problem remains **NP**-complete even if the input graph is bipartite [Garey and Johnson 1979].

The lack of efficient exact algorithms for general graphs has given rise to the possibility of finding approximation algorithms. In the case of dense graphs, an approximation of MinLA within a $1+\epsilon$ factor can be computed in time $n^{O(1/\epsilon)}$ for any $\epsilon > 0$ [Frieze and Kannan 1996]. The first non-trivial approximation algorithm for MinLA on general graphs was presented by [Even et al. 1995] and provides approximated solutions within a $O(\log n \log\log n)$ factor. To date, the best approximation

Class of graph	Complexity	Reference
Trees	$O(n^3)$	[Goldberg and Klipker 1976]
Rooted trees	$O(n \log n)$	[Adolphson and Hu 1973]
Trees	$O(n^{2.2})$	[Shiloach 1979]
Trees	$O(n^{\log 3/ \log 2})$	[Chung 1988]
Rectangular meshes	$O(n)$	[Muradyan and Piliposjan 1980]
Square meshes	$O(n)$	[Mitchison and Durbin 1986]
Hypercubes	$O(n)$	[Harper 1964]
de Bruijn graph of order 4	$O(n)$	[Harper 1970]
d-dimensional c-ary cliques	$O(n)$	[Nakano 1994]

Table 1. Survey of classes of graphs optimally solvable in polynomial time.

algorithm gives a $O(\log n)$ approximation factor for general graphs [Rao and Richa 1998]. However, that algorithm presents the disadvantage of having to solve a linear program with an exponential number of constraints using the Ellipsoid method. As a consequence, that algorithm is unsuitable for large graphs and very difficult to implement. Therefore, it is reasonable to investigate other heuristic methods that return good solutions in a reasonable amount of time. On the other hand, several methods to obtain lower bounds for MinLA are reported in [Juvan and Mohar 1992; Liu and Vannelli 1995].

The goal of this paper is to analyze different upper and lower bounding techniques for the MinLA problem. This is done running computational experiments on a selected testsuite of graphs. We focus on sparse graphs, for which no tight theoretical results exist. It is interesting to remark that the reduction used for proving the **NP**-completeness of this problem creates dense graphs [Garey et al. 1976], and that it is an open question to show if the decisional MinLA problem is still **NP**-complete when restricted to sparse graphs.

The paper is organized as follows. In Section 2, we present methods to find lower bounds for the MinLA problem. Several of the lower bounds have already been published but others are new. In Section 3, we present heuristics to approximate the MinLA problem. Several of these heuristics are adaptations of general heuristics (as Successive Augmentation and Local Search heuristics) for the MinLA problem. In Section 4 we briefly describe the LLSH toolkit that implements an interpreter from which it is possible to use the upper and lower bounding algorithms previously described. The architecture of this toolkit might be of interest to practitioners. Section 5 introduces the testsuite of graphs on which the experiments will be performed. This testsuite is made up of different families of graphs (arising both from "real life" graphs and from random models). In Section 6, we first present the environmental conditions of our experimental study. Then we present and discuss the results of several computational experiments aimed to analyze the different lower bounding methods and heuristics. The paper is concluded in Section 7 with a summary of our observations and is completed with an Appendix.

2. LOWER BOUNDS

Since the MinLA problem is **NP**-hard, it seems unlikely that any efficient algorithm to solve it can be found for general graphs. Therefore, it is difficult to evaluate the

performance of any heuristic by comparing its output with the optimum, except for a few classes of regular graphs described in the Introduction. This situation calls for seeking lower bounds to the cost of an optimal layout. In this section we review several known methods to obtain lower bounds for the MinLA problem as well as some new methods due to the author.

2.1 The Path method

The Path method was proposed by [Juvan and Mohar 1992]. Let $P_n^k = (V_n, E_n^k)$ denote the k-th power graph of the path graph with n vertices, where $V_n = \{1, \ldots, n\}$ and $E_n^k = \{ij \mid 0 < |i - j| \leq k\}$. It can be seen that

$$\text{MINLA}(P_n^k) = k(k+1)(3n - 2k - 1)/6.$$

Let $c(n, m)$ be the largest k for which $|E(P_n^k)| \leq m$; we have

$$c(n, m) = n - \tfrac{1}{2}\sqrt{(2n-1)^2 - 8m} - \tfrac{1}{2}.$$

In [Juvan and Mohar 1992] the following theorem is proved:

Theorem 1. *Let G a graph with n nodes and m edges. Then,* $\text{MINLA}(G) \geq \text{MINLA}(P_n^k)$, *where* $k = \lfloor c(n, m) \rfloor$.

2.2 The Edges method

Due to the floor operation taken in the previous theorem, the Path method ignores the length of some edges in the graph. A way to take them into account is to use the Edges method: Consider any layout φ. Notice that no more than $n - 1$ edges can have cost 1 in φ. Moreover, no more than $n - 2$ edges can have cost 2. In general, no more than $n - c$ edges can have cost c in any layout φ. This observation gives us a simple algorithm to compute a lower bound for the MinLA problem: while uncounted edges remain, count their minimal contribution:

```
function EdgesMethod (G : graph) : int
    n := |V(G)|; m := |E(G)|; i := 1; f := 0; lb := 0
    while f + n − i ≤ m :
        f := f + n − i
        lb := lb + i(n − i)
        i := i + 1
    return lb + i(m − f)
```

2.3 The Degree method

Consider any layout φ and define the contribution of a node u as $\sum_{uv \in E} |\varphi(u) - \varphi(v)|$. In the best case, a node must have two incident edges that contribute 1, two incident edges that contribute 2, two incident edges that contribute 3, etc. (see Figure 2). Therefore, a node u with degree d cannot have a contribution greater than $\sum_{i=1}^{d/2} 2i = d^2/4 + d/2$ if d is even or $\frac{1}{2}(d+1) + \sum_{i=1}^{(d-1)/2} 2i = (d^2 + 2d + 1)/4$ if d is odd. This remark gives another simple way to compute a lower bound for the MinLA problem: add the minimal contributions of each node and divide the sum by two (because edges have been counted twice).

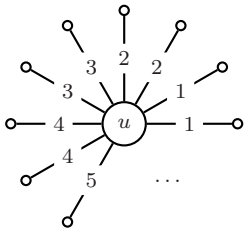


Fig. 2. Illustration for the Degree method.

function *DegreeMethod* (G : graph) : int
  $lb := 0$
  for all $u \in V(G)$:
    $d := \deg(u)$
    if $d \mod 2 = 0$ **then** $lb := lb + d^2/4 + d/2$ **else** $lb := lb + (d^2 + 2d + 1)/4$
  return $lb/2$

2.4 The Gomory–Hu Tree method

The Gomory–Hu Tree method was proposed by [Adolphson and Hu 1973]. Let $G = (V, E)$ be a graph with n nodes. Gomory and Hu showed that the adjacency maximal flow matrix f of G can simply be represented by a weighted tree $T = (V, E', w)$ where each edge $e \in E'$ represents a *fundamental cut* of G and has weight $w(e)$ equal to the corresponding minimal cut [Gomory and Hu 1975]. The maximum flow $f[i, j]$ between any pair of nodes (i, j) in G can be obtained by finding the minimal weight over all the edges in the path between i and j in T. Figure 3 shows a graph with its Gomory–Hu tree and its matrix f. The following theorem proved in [Adolphson and Hu 1973] gives a way to get a MinLA lower bound through the computation of a Gomory–Hu tree:

Theorem 2. *Let $G = (V, E)$ be a graph and $T = (V, E', w)$ its Gomory–Hu tree. Then,* $\text{LA}(G) \geq \sum_{e \in E'} w(e)$.

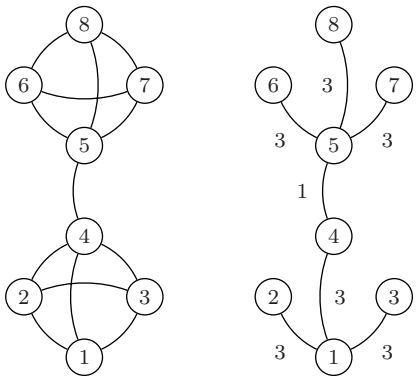
2.5 The Juvan–Mohar method

Let $G = (V, E)$ be a connected graph with $V = \{1, \ldots, n\}$ and L_G its $n \times n$ Laplacian matrix defined as follows for all $u, v \in V$:

$$L_G[u, v] = \begin{cases} 0 & uv \notin E, \\ -1 & uv \in E, \\ \deg(u) & u = v. \end{cases}$$

The smallest eigenvalue of L_G is zero (because L_G is positive semidefinite). Let λ_2 be the second smallest eigenvalue of L_G. This eigenvalue gives a measure of the connectivity of the graph. The following theorem is proved in [Juvan and Mohar 1992]:

Theorem 3. *Let λ_2 be the second smallest eigenvalue of the Laplacian matrix of a connected graph G. Then,* $\text{MINLA}(G) \geq \lambda_2 \left(n^2 - 1\right)/6$.



f	1	2	3	4	5	6	7	8
1		3	3	3	1	1	1	1
2	3		3	3	1	1	1	1
3	3	3		3	1	1	1	1
4	3	3	3		1	1	1	1
5	1	1	1	1		3	3	3
6	1	1	1	1	3		3	3
7	1	1	1	1	3	3		3
8	1	1	1	1	3	3	3	

Fig. 3. A graph G, its Gomory–Hu tree and its matrix f of max-flows min-cuts.

2.6 The Mesh method

Let L_n be a square mesh of side n, that is, $V(L_n) = \{1, \ldots, n\}^2$ and $E(V_n) = \{uv \mid \|u - v\|_2 = 1\}$. [Muradyan and Piliposjan 1980] (and latter [Mitchison and Durbin 1986]) proved that $\text{MINLA}(L_n) = (4 - \sqrt{2})n^3/3 + O(n^2)$. Therefore, another way to obtain a lower bound for the MinLA problem is to decompose an input graph G into t disjoint square meshes $M_1, \ldots, M_t$, because $\text{MINLA}(G) \geq \sum_{i=1}^{t} \text{MINLA}(M_i)$ and $\text{MINLA}(M_i)$ is known.

We propose the following greedy algorithm to compute a lower bound for MinLA by iteratively finding maximal square meshes in a graph:

function *MeshMethod*(G) : int
  $lb := 0$
  Let M be the largest square mesh contained in G; let s be its side
  while $s \geq 2$:
    $lb := lb + (4 - \sqrt{2})s^3$
    $G := G \setminus M$
    Let M be the largest square mesh contained in G; let s be its side
  return lb + *EdgesMethod*(G)

In order to find the largest square mesh contained in G, it is necessary to resort to backtracking, which renders this method quite inefficient (although a careful implementation can prune much of the search space).

2.7 Discussion

In the case of sparse graphs where $|E| = O(|V|)$, it can be noticed that the lower bounds obtained by the Path method, the Edges method and the Degree method are linear in $|V|$. This fact shows that these methods might not be very sharp. In contrast, the lower bounds obtained by the Juvan–Mohar method can grow faster than linearly, depending on the expansion properties of the input graphs. It also can be expected that the bounds obtained by the Mesh Method can be effective in graphs raising from applications in finite element methods, because these graphs contain many large sub-meshes.

3. APPROXIMATION HEURISTICS

In this section we present several heuristics to get approximate solutions for the MinLA problem.

3.1 Random and Normal layouts

A simple way to generate approximated solutions consists in returning a random feasible solution, i.e., a *random layout*. A similar idea consists in not permuting at all the input (recall that we consider graphs whose nodes are labelled). This is what we call the *normal layout*:

$$\varphi(i) = i, \qquad \forall i \in \{1, \ldots, n\}.$$

Of course, these methods will yield bad results in general, but at least their running time will be negligible.

3.2 Successive Augmentation heuristics

We present now a family of *Successive Augmentation heuristics*. In this approach, a partial layout is extended, vertex by vertex, until all vertices have been enumerated, at which point the layout is output without any further attempt to improve it. At each step, the best possible free label is assigned to the current vertex. These types of heuristics have been applied to a great variety of optimization problems, such as the Graph Coloring problem or the Traveling Salesperson problem [Johnson 1990; Johnson et al. 1991].

Our generic heuristic works as follows (see algorithm below). To start, label 0 is assigned to an arbitrary vertex. Then, at each iteration, a new vertex is added to the layout, to its left or to its right, according to the way that minimizes the partial cost. The layout will be from $l+1$ to $r-1$ and not from 1 to i as usual, but this has no importance, because MinLA works with differences between labels and not with the labels them-selves. In order to decide in which extreme of the current layout the new vertex must be placed, a function $\mathit{Increment}(G, \varphi, i, v_i, x)$ returns the increment of the cost of the partial layout φ restricted to the vertices $v_1, \ldots, v_{i-1}$ when label x is assigned to vertex v_i. Finally, a second loop maps φ to $\{1, \ldots, n\}$.

function *Increment* (G, φ, i, v_i, x) : int
  $\varphi[v_i] := x;\ c := 0$
  for $j := 1$ **to** i :
    if $v_i v_j \in E$:
      $c := c + |\varphi[v_i] - \varphi[v_j]|$
  return c

function *SuccessiveAugmentation* (G) : layout
  $n := |V(G)|$
  Select an initial ordering of vertices $v_1, v_2, \ldots, v_n$
  $\varphi[v_1] := 0;\ l := -1;\ r := 1$
  for $i := 2$ **to** n :
    if $\mathit{Increment}(G, \varphi, i, v_i, l) < \mathit{Increment}(G, \varphi, i, v_i, r)$:
      $\varphi[v_i] := l;\ l := l - 1$ ▷ Put at left

```
    else
        φ[v_i] := r;  r := r + 1      ▷ Put at right
▷ Remap φ to {1, ..., n}
for i := 1 to n :  φ[i] := φ[i] − l
return φ
```

Let us now turn the discussion to the initial ordering of the vertices. We propose four different strategies:

—Normal ordering: The vertices are ordered in the same way as they are labelled in the graph.

—Random ordering: The vertices are randomly ordered. This scheme has the disadvantage of ignoring the connectivity and density of the graph.

—Random breadth search: Choose an initial vertex v_1 and let $S := \{v_1\}$ and $i := 2$. While $S \neq V$, choose randomly an edge $uv_i \in E$ with $u \in S$ and $v_i \notin S$; add v_i to S letting $S := S \cup \{v_i\}$ and increment i. This initial ordering has the advantage of making use of the connectivity of the graph, but it lacks locality.

—Breadth-first search: To create an initial ordering, perform a breadth-first search from a random node of the graph. In this way, the greedy heuristic would take profit of the possible locality and connectivity of the graph.

—Depth-first search: An alternative to the previous strategies would be to perform a depth-first search from a random node of the graph.

If the input graph is stored using sorted adjacency lists (for each vertex a sorted list of its neighbours is stored), the family of Successive Augmentation heuristics that we have proposed have complexity $O(n^2 \log n)$, which dominates the cost of the initial ordering of the vertices. Observe that all its variations (except the normal ordering) are randomized algorithms because, at least, the output depends on the first chosen vertex. Furthermore, ties might be broken by random decisions.

3.3 Spectral Sequencing

We review now a heuristic to find layouts original from [Juvan and Mohar 1992]. Given a graph G, the algorithm first computes the *Fiedler vector* of G; that is, the eigenvector $x^{(2)}$ corresponding to the second smallest eigenvalue λ_2 of the Laplacian matrix L_G of G. Then, each position of $x^{(2)}$ is ranked. Thus, the heuristic returns a layout φ satisfying

$$\varphi(u) \leq \varphi(v) \quad \text{whenever} \quad x_u^{(2)} \leq x_v^{(2)}.$$

The rationale behind this heuristic is that the ordering of the vertices produced by their values in the Fiedler vector has some nice properties. In particular, vertices connected by an edge will tend to be assigned numbers that are close to each other. This property has been used already in other problems such as graph partitioning, chromosome mapping or matrix reordering [Atkins et al. 1999; Hendrickson and Leland 1993].

3.4 Local Search heuristics

Local Search has been described as "an area where intuition and empirical tests play a crucial role and where the design of effective Local Search is much an art" [Pa-

padimitrou and Steiglitz 1982]. In spite of this, because of its performance and simplicity, local search is one of the most used techniques to approximate many combinatorial problems. The basic principle of this heuristic is to iteratively improve a randomly generated solution by performing local changes on its combinatorial structure. Usually, changes that improve the solution are accepted, whereas that changes that worsen the solution are rejected.

In order to apply Local Search to a minimization problem, the following items should be recognized: the set of feasible solutions ($S = \{\sigma_i\}$), a cost function that assigns a numerical value to any feasible solution ($f : S \rightarrow \mathbb{R}^+$), and a neighborhood, which is a relation between feasible solutions that are "close" in some sense. The generic skeleton (subject to many variations) is as follows:

function *LocalSearch* ( )
  $\sigma :=$ Select initial random feasible solution
  while $\neg$*Termination*() :
    $\sigma' :=$ Select a neighbor of σ
    $\delta := f(\sigma) - f(\sigma')$
    if *Acceptable*(δ) :
      $\sigma := \sigma'$
  return σ

For the MinLA problem, the set of all feasible solutions is the set of all layouts (permutations) of size n, and the objective function is $\text{LA}(G, \varphi)$. However many different neighborhoods can be taken into account; in this work we have considered the following:

—Flip2: Two layouts are neighbors if one can go from one to the other by flipping the labels of any pair of nodes in the graph.

—Flip3: Two layouts are neighbors if one can go from one to the other by rotating the labels of any trio of nodes in the graph.

—FlipE: Two layouts are neighbors if one can go from one to the other by flipping the labels of two adjacent nodes in the graph.

Besides the appealing simplicity of these neighborhoods, the reasons to choose them among all the other potential candidates are the easiness to perform movements and the low effort necessary to compute incrementally the cost of the new layout.

We remark that any move in the FlipE neighborhood can also be produced in the Flip2 neighborhood, and that any move in the Flip2 neighborhood can also be produced in the Flip3 neighborhood. It is uncertain which one of these neighborhoods is more suitable, because even though Flip3 will probably not stop in many local minima, as its size is bigger than Flip2 or FlipE, its exploration consumes more time.

Figure 4 shows a graphical representation of the landscape defined by the Flip2 neighborhood. For a random graph with 20 vertices, this figure shows the magnitude of the gain of swapping any pair of vertices. It can be remarked that the landscape is very irregular.

Below, we present some heuristics derived from the general local search skeleton. These variations depend on the way the neighborhood is explored to search favor-

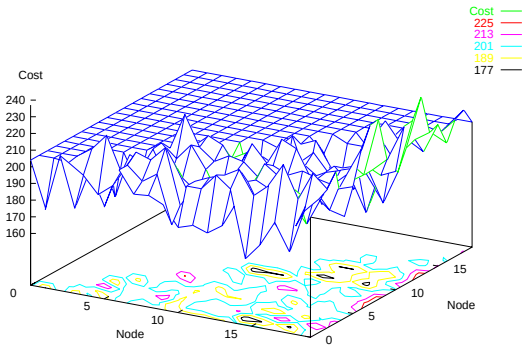


Fig. 4. Landscape of Flip2 on a random graph.

able moves or which is the criterion to accept moves that do not directly improve the solution. These algorithms are usually called *black-box heuristics* [Juels 1996] because they work only with the objective function and the neighborhood structure, but do not use problem-dependent strategies.

3.5 Hillclimbing

The hillclimbing heuristic on the Flip2 neighborhood has been implemented as follows: A first initial layout is generated at random. Then, proposed moves are generated at random and are accepted when their gain (δ) is positive or zero (in order to go across plateaus). The heuristic terminates after max consecutive proposed moves have not strictly decremented the cost of the layout. The algorithm for Hillclimbing is given below. The function $GainWhenFlip(G, \varphi, u, v)$ returns the gain (or decrease) in the cost function when flipping the labels of u and v in φ, and the procedure $Flip2(\varphi, u, v)$ performs this flipping.

function *HillClimbing2* (G : graph, max : int) : layout
  $\varphi :=$ Generate an initial random layout; $z := 0$
  repeat
    $z := z + 1$
    $u :=$ Generate a random integer in $\{1, \ldots, n\}$
    $v :=$ Generate a random integer in $\{1, \ldots, n\}$
    $\delta := GainWhenFlip2(G, \varphi, u, v)$
    if $\delta \geq 0$:
      $Flip2(\varphi, u, v)$
      if $\delta > 0$: $z := 0$
  until $z = max$
  return φ

The hillclimbing heuristic with the Flip3 neighborhood proceeds in the same way, except that three nodes are randomly chosen. For the FlipE neighborhood, first a node u is randomly chosen among V, then another node adjacent to u is randomly chosen. On the Flip2 and Flip3 neighborhoods, we took $max = n \log_2 n$; on FlipE, $max = \log_2^2 n$.

3.6 Full search

In the full search heuristic, at each step the gain of each possible transition is computed in order to choose the move with maximum gain in the current neighborhood. The algorithm on the Flip2 neighborhood is conceptually as follows:

```
function FullSearch (G : graph, max : int) : layout
    φ := Generate an initial random layout; z := 0
    repeat
        z := z + 1
        ⟨u, v, δ⟩ := SelectBestMove(G, φ)
        if δ ≥ 0 :
            Flip2(φ, u, v)
            if δ > 0 : z := 0
    until z = max
    return φ
```

According to this implementation, it seems necessary to compute at each steep the gain of the $n(n-1)/2$ possibles moves. However, large savings can be done if the graph is sparse, because it is not necessary to recompute the moves of the nodes that are not neighbors of the previously interchanged nodes. In this way, the cost of each iteration (except the first one) is reduced to $O(dn)$ where d is the maximum degree of the input graph.

3.7 The Metropolis heuristic

The problem of all Local Search heuristic we have seen so far is that once a local optimum is found the heuristic stops, but this local optimum can be far away from the global optimum. In order to enable the heuristic to accept downhill moves, the Metropolis heuristic [Metropolis et al. 1953] is parametrized by a temperature t and proceeds as follows (using the Flip2 neighborhood):

```
function Metropolis (G : graph, r : int) : layout
    φ := Generate an initial random layout
    for i := 1 to r :
        u := Generate a random integer in {1, ..., n}
        v := Generate a random integer in {1, ..., n}
        δ := GainWhenFlip2(G, φ, u, v)
        with probability min(1, e^(−δ/t)) :
            Flip2(φ, u, v)
    return φ
```

Observe that uphill movements will be automatically accepted, whereas downhill movements are accepted randomly in function of the "height" (δ) of the movement and the temperature t. With a high temperature the probability of descending is high; with a small temperature, it is low. In the limit, as $t \to \infty$ Metropolis performs a random walk on the neighborhood structure and as $t \to 0$ Metropolis proceeds as the hillclimbing heuristic.

3.8 Simulated Annealing

The Simulated Annealing heuristic [Kirkpatrik et al. 1983] is closely related to the Metropolis process. Briefly, SA consists of a sequence of runs of Metropolis with progressive decrement of the temperature (the t parameter). For the MinLA problem, the basic Simulated Annealing algorithm follows the following algorithm:

function *SimulatedAnnealing* (G : graph) : layout
  $\varphi :=$ Generate an initial random layout
  $t :=$ *InitialTemperature()*
  while $\neg$*Frozen()* :
    while $\neg$*Equilibrium()* :
      $u :=$ Generate a random integer in $\{1, \ldots, n\}$
      $v :=$ Generate a random integer in $\{1, \ldots, n\}$
      $\delta :=$ *GainWhenFlip2*(G, φ, u, v)
      with probability $\min(1, e^{-\delta/t})$:
        Flip2(φ, u, v)
    $t := \alpha \cdot t$ $\triangleright\ 0 < \alpha < 1$
  return φ

The main point in SA algorithms is the selection of their parameters (initial temperature, frozing and equilibrium detection, cooling ratio $\alpha \ldots$). These depend not only on the optimization problem (MinLA in our case) but also on the instance of the problem. An excellent treatment of different SA schemes can be found in [Johnson 1990; Johnson et al. 1989; Johnson et al. 1991]. Rather than investigating different selections for these parameters, we have used an adaptative technique due to Aarts [van Laarhoven and Aarts 1987].

4. THE LLSS TOOLKIT FOR THE MINLA PROBLEM

All the upper and lower bounding techniques described in this paper have been implemented in order to enable their evaluation. The result is the "LLSH toolkit", a library of methods for the MinLA problem, which we briefly describe in this section.

The architecture of this toolkit contains two layers. In the core layer resides the implementation of the different upper and lower bounding methods, together with utilities to treat layouts and manipulate graphs. This core is implemented in C++. The toolkit is completed with an interface layer which drives the core using a Tcl interpreter [Ousterhout 1994]. This architecture proves to be very convenient: on one side, the fact that the core is implemented in C++ enables both a high level and efficient implementation of the different methods; on the other side, offering an interpreter to the users of the toolkit enables an easy and fast coding of the driver programs that perform the experiments and process the results.

The toolkit core was developed with efficiency as a primary goal. In order to reduce the programming effort, several preexisting libraries have been used:

—In order to quickly compute the Fiedler vector of a large sparse matrix with precision and without taking too many resources, LLSH uses an implementation of the Lanczos algorithm [Parlett et al. 1982] offered in the Chaco library [Hendrickson and Leland 1995].

`llsh`	Starts the LLSH toolkit
`% Graph G "randomA1.gra"`	Creates a graph `G` loading `randomA1`
`% puts "[G nodes] [G edges]"`	Writes the number of nodes and edges of `G`
`1000 4974`	result computed by LLSH
`% Layout L [@ G]`	Creates a layout `L` for the graph `G`
`% UB_spectral [@ G] [@ L]`	Applies the spectral method to `G` and sets the result in `L`
`% puts [L la]`	Writes the cost of the layout `L`
`1202165`	result computed by LLSH
`% puts [LB_juvan_mohar [@ G]`	Writes the Juvan–Mohar lower bound for `G`
`140634`	result computed by LLSH
`% exit`	Leaves LLSH

Fig. 5. Sample session with LLSH.

—In order to compute the Gomory–Hu tree of a graph, a public domain implementation done by [Skorobohatyj 1994] has been ported to C++.

—The simulated annealing heuristic has been implemented using the parSA library [Klohs 1999], which offers the Aarts adaptative scheduler.

In order to illustrate how the interpreter of the toolkit works, Figure 5 shows a sample session.

5. A TEST SUITE FOR EVALUATING MINLA METHODS

We consider now the creation of a test suite of graphs to benchmark the previous upper and lower bounding techniques for the MinLA problem.

There exist several test suites for various combinatorial optimization problems. For instance, TSPLIB is a library of sample instances for the TSP from various sources and of various types [Reinelt 1991]. Also, QAPLIB provides a unified test bed for the Quadratic Assignment problem [Burkard et al. 1997]. Both libraries are regularly used by practitioners to test new algorithms, and owe their utility to be accessible to the scientific community through the World Wide Web. In contrast, no test suite has already been proposed for MinLA.

Creating a test suite of graphs for the MinLA problem is a difficult task. Ideally, this test suite should contain graphs arising from real life applications, graphs with known optima, graphs that can support general conclusions, and generators of instances. Unfortunately, these kinds of graphs do not abound. Therefore, in order to select the graphs in the test suite, some pragmatical decisions must be taken:

—The first decision we take is to limit the scope of the test suite to sparse graphs. It has already been said that for the case of dense graphs, fully approximation schemes exist, so this family of graphs has not much interest for us.

—The second decision we take is to only include large graphs in the test suite. Here, "large" should stand for graphs that cannot be optimally solved by a brute force algorithm as Branch and Bound in a "reasonable" time. This is a vague definition, but has the advantage of being useful and dynamical. It can also encourage the study of brute force algorithms to show that, after all, these graphs were not so large. Currently, according to this definition, a 5×5 square grid graph is large. However, during this research we have felt the need to go further, and so we have

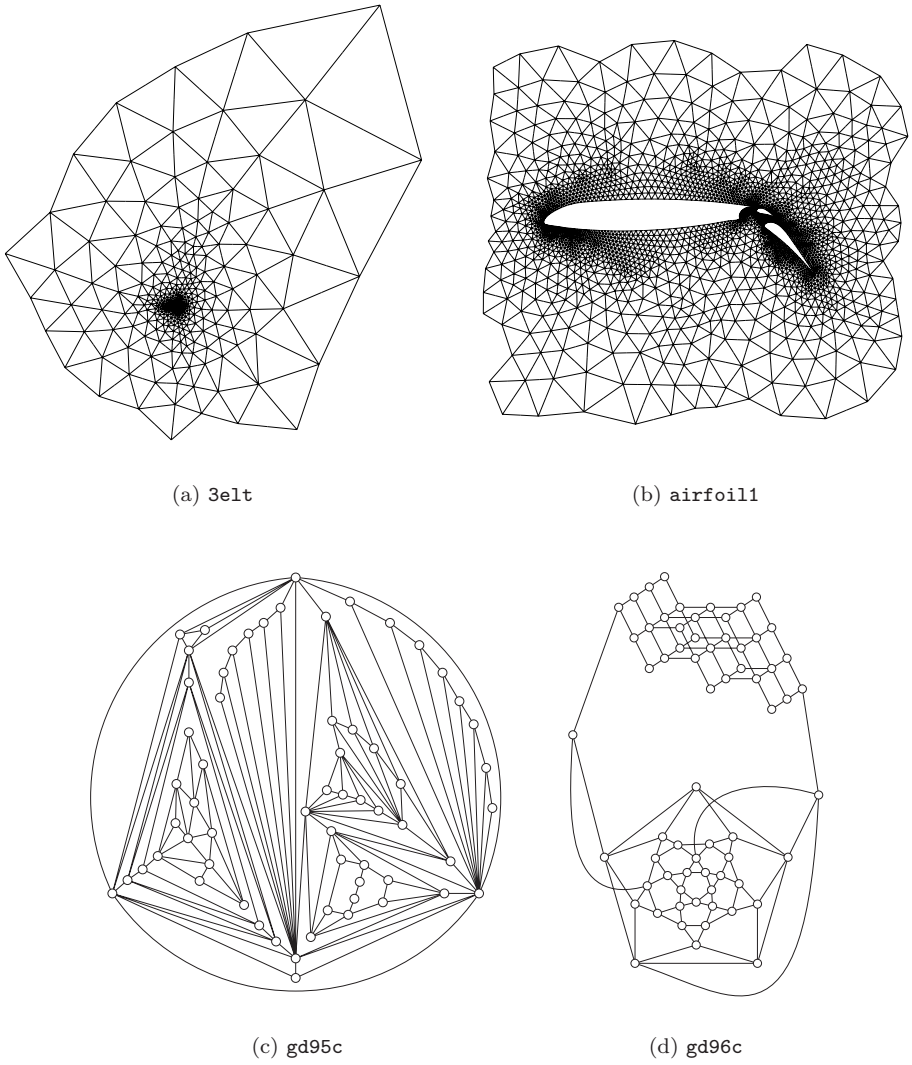
decided that the graphs to be included in the test suite should have more than one thousand vertices.

—The third decision we take is that the test suite must contain several graphs for which their optimal solution is known. At the current time, the only families of graphs that satisfy this condition are the ones shown in Table 1. From these graphs, we select a binary tree, a hypercube and a square grid. These are natural families of graphs and represent quite different topologies. In order to have graphs with around one thousand vertices, we take a 10-hypercube (`hc10`), a 33×33 grid graph (`mesh33x33`) and a complete binary tree with 10 levels (`bintree10`).

—The fourth decision we take is that random graphs should be included in the test suite. The reason is that such graphs may be amenable to a probabilistic analysis, and thus general conclusions can be drawn from their study.
Binomial random graphs $\mathcal{G}_{n,p}$ are obvious candidates to be included in the test suite. Recall that a $\mathcal{G}_{n,p}$ graph is a graph with n vertices where each of the $n(n-1)/2$ edges appears with probability p. We include two $\mathcal{G}_{n,p}$ binomial random graphs in the test suite, taking $n = 1000$ and $p = 0.01$ and $p = 0.05$. The random graphs generated are `randomA1` and `randomA2`.
Besides $\mathcal{G}_{n,p}$ graphs, it would be interesting to include some other class of random graphs. Random regular graphs do not seem an option, as they share many properties with binomial random graphs, which already have been included. On the other hand, random geometric graphs seem a good option. Random geometric graphs are graphs whose vertices correspond to points uniformly distributed at random in the unit square, and whose edges join vertices that are closer than some specified bound. Random geometric graphs have also often been used as instances to benchmark heuristics, see for example [Berry and Goldberg 1999; Johnson et al. 1989; Lang and Rao 1993]. So, the test suite includes a random geometric graph, `randomG4`, with 1000 vertices and radius 0.075. Finally, the inclusion of a random geometric graph calls for the inclusion of a binomial random graph with a similar number of vertices and edges. The graph `randomA4` was therefore generated in order to discover differences with `randomG4`.

—The fifth and last decision we take is to include "real life graphs" in the test suite. As we could not obtain real life instances specific for MinLA, we have taken three families of graphs that arise in the same kinds of applications that layout problems: VLSI design (VLSI family), graphs from finite element discretizations (FE family) and graphs from graph-drawing competitions (GD family). The VLSI graphs `c1y`, `c2y`, `c3y`, `c4y` and `c5y` are graphs derived from circuit layouts. The FE graphs arise from applications in computational fluid dynamics (`airfoil1` and `3elt`), earthquake wave propagation (`whitaker3`) and structural mechanics (`crack`). Finally, the GD graphs have been obtained from several graph-drawing competitions posted in the World Wide Web. In this class, `gd95c` is a mysterious graph, `gd96a` represents a finite automaton used in a natural-language processing system, `gd96b` represents the calls made between a set of telephone numbers, `gd96c` is an artificial nice graph, and `gd96d` represents the structure of a fragment of a web site.

The main characteristics of the graphs in our test suite are shown in Table 2. Some of them are shown in Figure 6.



(a) `3elt` (b) `airfoil1`

(c) `gd95c` (d) `gd96c`

Fig. 6. Some graphs from the test suite.

Name	Nodes	Edges	Degree	Diam	Family
randomA1	1000	4974	1/9.95/21	6	$\mathcal{G}_{n=1000,p=0.01}$
randomA2	1000	24738	28/49.47/72	3	$\mathcal{G}_{n=1000,p=0.05}$
randomA4	1000	8177	4/16.35/29	4	$\mathcal{G}_{n=1000,p=0.0164}$
randomG4	1000	8173	5/16.34/31	23	$\mathcal{G}_{n=1000}(r = 0.075)$
hc10	1024	5120	10/10/10	10	10-hypercube
mesh33x33	1089	2112	2/3.88/4	64	33×33-mesh
bintree10	1023	1022	1/1.99/3	18	10-bintree
3elt	4720	13722	3/5.81/9	65	FE
airfoil1	4253	12289	3/5.78/10	65	
crack	10240	30380	3/5.93/9.00	121	
whitaker3	9800	28989	3/5.91/8	161	
c1y	828	1749	2/4.22/304	10	VLSI
c2y	980	2102	1/4.29/327	11	
c3y	1327	2844	1/4.29/364	13	
c4y	1366	2915	1/4.26/309	14	
c5y	1202	2557	1/4.25/323	13	
gd95c	62	144	2/4.65/15	11	GD
gd96a	1076	1676	1/3.06/111	20	
gd96b	111	193	2/3.47/47	18	
gd96c	65	125	2/3.84/6	10	
gd96d	180	228	1/2.53/27	8	

Table 2. The testsuite. For each graph, its name, number of nodes, number of edges, degree information (minimum/average/maximum), diameter and family.

6. EXPERIMENTAL EVALUATION

In this section we present, compare and analyze some experimental results aiming to evaluate the performance of the considered methods.

6.1 Experimental environment

As described in Section 4, the core of the programs has been written in the C++ programming language, and the drivers that perform the experiments have been written in Tcl. The experiments have been run on a cluster of identical PCs with AMD-K6 processors at 450 MHz and 256 Mb of memory with a Linux operating system delivering 715.82 BogoMips. These computers have enough main memory to run our programs without delays due to paging. Moreover, all the experiments have been executed in dedicated machines (except for system daemons) and measure the total elapsed (wall-clock) time. Pre- and post-processing times are included in our measures (except for the time used to read the input graph). The programs have been compiled using GCC 2.95.2 with the `-O3` option.

Even if the cluster allows us to run parallel programs, we have not done so because we have preferred to run many independent runs of sequential programs on different processors simultaneously. Performing a large number of runs has been necessary because of the randomized nature of the algorithms, which yield different values and execution times from run to run.

The code has originally been tested using two different random number genera-

Graph	Edges	Degree	Path	J–M	G–H	Mesh	best ub
randomA1	14890	16176	9970	140634	9926	†	884261
randomA2	321113	323568	319475	4429294	49404	†	6528780
randomA4	37713	39531	35796	601130	16325	†	1721670
randomG4	37677	39972	35796	14667	16315	†	146996
bintree10	1022	1277	1022	173	1022	1	*3696
hc10	15395	15360	15305	349525	10230	32768	*523776
mesh33x33	3136	3135	1088	1789	4220	*31680	*31680
3elt	27010	27135	14155	8476	27435	44785	363204
airfoil1	24112	24220	12754	5571	24569	40221	277412
crack	60424	64938	30715	25826	60751	95347	1491126
whitaker3	57571	57824	29395	11611	57970	144854	1151064
c1y	2767	14101	2479	13437	3192	2819	62233
c2y	3370	16473	2935	17842	3877	3762	79429
c3y	4555	20874	3976	23417	5320	5548	123548
c4y	4651	16404	4093	21140	5518	5778	115222
c5y	4069	16935	3601	19217	4790	4626	96965
gd95c	250	292	181	36	255	174	506
gd96a	2257	4552	1095	5155	3233	377	96342
gd96b	276	702	110	43	305	113	1422
gd96c	186	191	64	38	241	130	519
gd96d	277	595	179	415	331	113	2409

† The Mesh method could not be applied
* Optimal
Best lowerbounds underlined

Table 3. Comparison of the lower bounds methods.

tors (the standard `rand()` function in `<stdlib.h>` and the one in LEDA) without noticing any anomaly due to them.

All the elemental experiments have been executed independently 200 times, except for Simulated Annealing (from 5 to 25 independent runs depending on the graph size).

In order to help to reproduce and verify the measurements and the code mentioned in this research, its code, instances and raw data are available on the Internet at http://www.lsi.upc.es/∼jpetit/MinLA/Experiments.

6.2 Comparison of the lower bounds methods

In order to compare the different lower bounding methods presented in Section 2, we have applied each method to get a lower bound for each graph included in our test suite. The results, together with the best approximations found in this research, are shown in Table 3.

The first fact that can be observed in Table 3 is that the highest obtained lower bounds are far away from the best observed upper bounds; the only exception is the case of the `mesh33x33` graph, for which the optimal lower bound is obviously reached by the Mesh method.

The comparison between the different lower bounds makes clear that all the

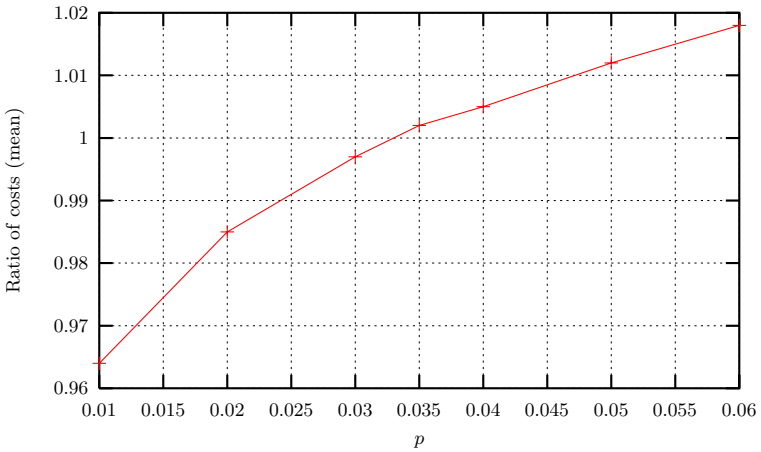


Fig. 7. Comparison between the Flip2 and Flip3 neighborhoods with the Hillclimbing algorithm on $\mathcal{G}_{n=1000,p}$ graphs. The curve shows the ratio between the average results obtained using Flip2 and Flip3 on 100 independent runs.

methods have different behaviors depending on the class of graph they are applied to: 1) for the uniform random graphs, the Juvan–Mohar bound gives much better results than any other method; 2) in the case of FE graphs, the Mesh method always delivers the best lower bounds; 3) on our VLSI class of graphs, the Juvan-Mohar method and the the Degree method clearly outperform the other methods; 4) on the random geometric graph, the Degree method obtains the best lower bound.

As expected, the Edges method always dominates the Path method. Furthermore, we can observe that the Gomory–Hu tree method and the Edges method provide bounds of similar quality, which are always dominated by the Degree method. An interpretation of the poor performance of the Gomory–Hu tree method is that, for all the graphs in our testsuite, their Gomory–Hu tree is a star whose central node has maximal degree.

As for the running times, the Edges, Degree and Path methods have a negligible running time; computing the Juvan–Mohar bound takes less than 5 seconds and computing the Gomory–Hu tree takes about 10 minutes on our bigger graphs. Case apart is the Mesh method, for which we had to limit its exploration to meshes up to 23×23 nodes in order to get the results on the FE graphs. In spite of this limit, the Mesh method can last 14 hours on the `whitaker3` graph and lasts for days on the random graphs.

From these results, the best advice to obtain the best lower bounds for the MinLA problem on large sparse graphs is to use the Juvan–Mohar bound and the Degree method, which provide good results, are quite fast and always dominate the Path, Edges and Gomory–Hu methods. Therefore, applying the Mesh method is only worthwhile in the case of graphs with large submeshes, provided one can afford the long running time needed to compute it. In any case, the experiments evidence that none of the presented lower bounding techniques might be very tight in general.

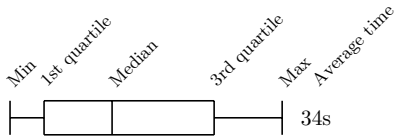


Fig. 8. Interpretation of the boxplots.

Legend	Meaning
spectral	Spectral Sequencing
sa	Simulated Annealing
random	Random method
hillc_E	Hillclimbing using the FlipE neighborhood
hillc_3	Hillclimbing using the Flip3 neighborhood
hillc_2	Hillclimbing using the Flip2 neighborhood
greedy_rnd	Successive Augmentation method with random initial ordering
greedy_rbs	Successive Augmentation method with random breadth search
greedy_nor	Successive Augmentation method with normal ordering
greedy_dfs	Successive Augmentation method with depth-first search
greedy_bfs	Successive Augmentation method with breadth-first search
normal	Normal method
best ub	Best upper bound found in this research
Edges	Edges method
Degree	Degree method
Path	Path method
J–M	Juvan–Mohar method
G–H	Gomory–Hu Tree method
Mesh	Mesh method

Table 4. Abreviations used in figures and tables.

6.3 Comparing the Flip2 and Flip3 neighborhoods

Notice that when using full exploration, the Flip3 neighborhood will be stuck at fewer local minima than the Flip2 neighborhood. However, in the case of Hillclimbing algorithm things are not so evident. Moreover, in Flip3 it is more difficult to find good moves than in Flip2. In preliminary tests, it seemed that the Flip2 neighborhood was working better than the Flip3 neighborhood for very sparse graphs.

To validate this first impression, we analyzed the performance of Hillclimbing on random graphs $\mathcal{G}_{n,p}$ for $n = 1000$ and $0.01 \leq p \leq 0.06$ using the two types of neighborhoods. The average result of performing this experiment 100 times is shown in Figure 7. Recall that in both cases we perform the same number of iterations.

As Figure 7 reveals, there is a relation between the ratio of the results of the two algorithms and the edge probability p: in the case that $p \leq 0.03$, the Flip2 neighborhood yields slightly better results; in the case that $p \geq 0.035$, the Flip3 neighborhood turns out to be better.

6.4 Graphs with known minima

In the case of graphs where the optimal MINLA value is known, we can use those optimal values to measure the quality of the results obtained by the heuristics we have described. The results of applying each heuristic to these graphs are reflected in Figure 9, which shows the performance of the heuristics both in approximation quality and running time.

Table 4 resumes the abbreviations shown in the figures and tables. The results are summarized using boxplots. A *boxplot* is a graphical way of summarizing the distribution of the values of a group of samples; see Figure 8. By portraying the values for more than one group next to each other, one can see the major trends in the dataset. The box in a boxplot shows the median value as a line and the first (25th percentile) and third quartile (75th percentile) of the value distribution as the lower and upper parts of the box. The bars shown at the sides of the boxes represent the largest and smallest observed values. The average time elapsed to compute an individual of the sample is displayed at the right of the corresponding boxplot. For the sake of clarity, absolute values have been normalized by the optimal values, given in Table 3.

All the graphs share in common that the best average results are found by Simulated Annealing. In the case of the hypercube (`hc10`), Simulated Annealing hits the optimum in 25% of the tries; for the mesh (`mesh33x33`), Simulated Annealing finds solutions not exceding 44% of the optimum; for the binary tree (`bintree10`) the solutions obtained by Simulated Annealing almost double the optimal value.

Depending on the graph, some methods work better than others. In any case Simulated Annealing always dominates them, at the cost of having a longer running time. It must be noticed that on the hypercube, the Normal and Succesive Augmentation heuristics (with normal ordering) reach the optimal values simply because the MINLA of a hypercube is reached on the normal numbering.

6.5 Uniform random graphs versus geometric random graphs

It is clear that uniform random graphs $\mathcal{G}_{n,p}$ and geometric random graphs $\mathcal{G}_n(d)$ have very different characteristics. For instance, the graphs `randomA4` and `randomG4`

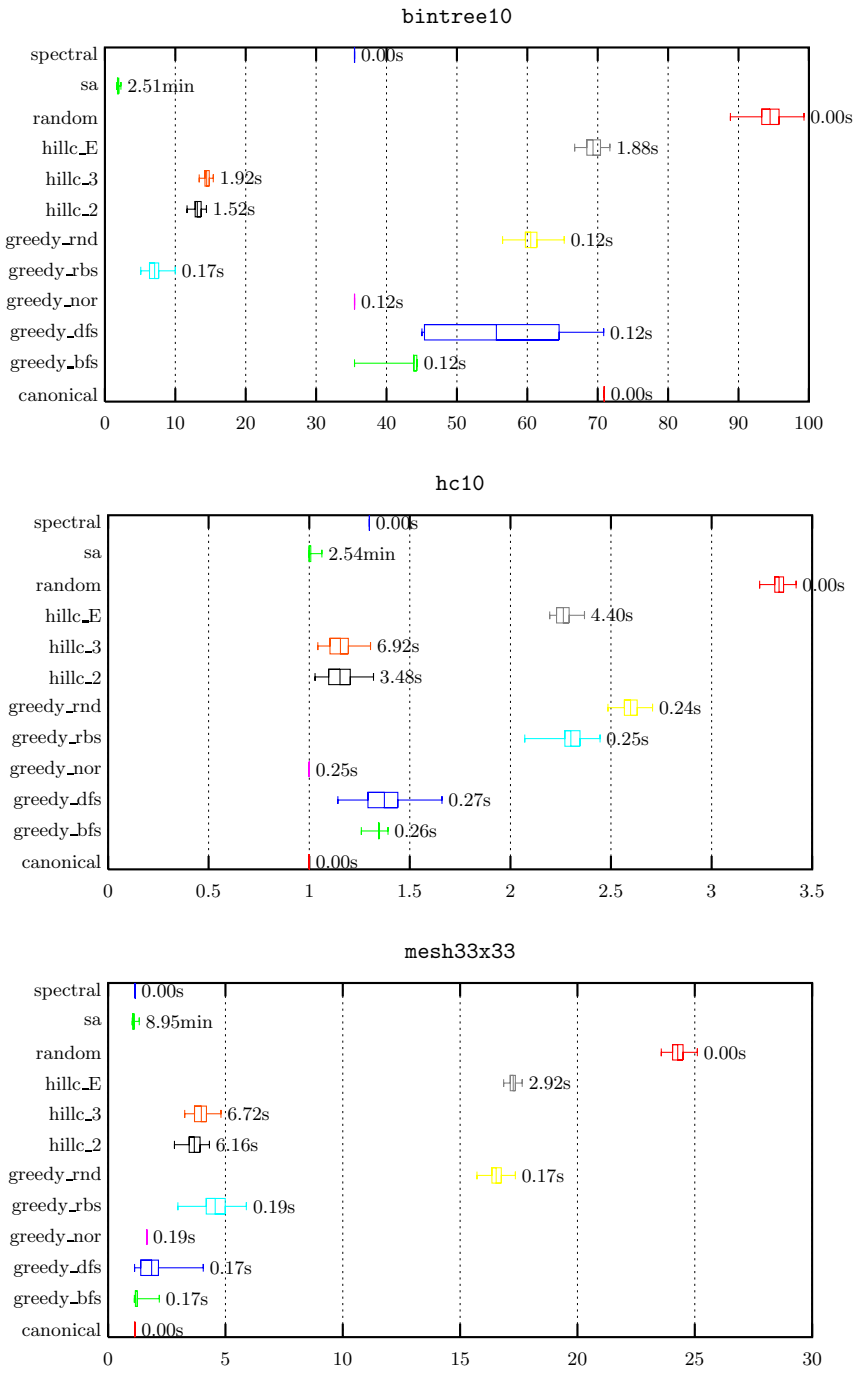


Fig. 9. Comparison of heuristics for graphs with known optima. The absolute costs have been normalized by dividing them by the respective optimal costs.

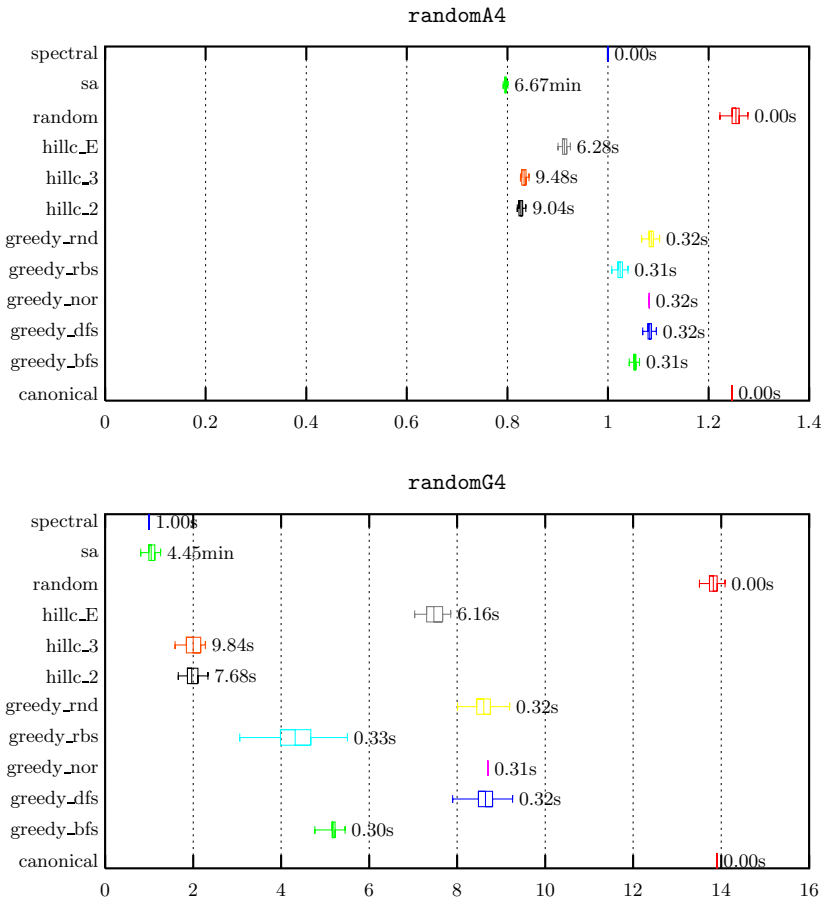


Fig. 10. Comparison of heuristics for a uniform random graph and a random geometric graph with similar number of edges and nodes. The absolute costs have been normalized dividing them by the cost found by Spectral Sequencing.

have the same numbers of nodes and edges (nearly), but a very different diameter (see Table 2). How do these inherent properties of different classes of random graphs affect the behavior of the heuristics we have presented? In order to answer this question, each analyzed heuristic has been applied to the `randomA4` and `randomG4` graphs. The results are shown in Figure 10.

In the uniform random graph, the best solutions are obtained with the Local Search heuristics. Simulated Annealing obtains the best costs, but the costs obtained with Hillclimbing are very close. The solutions obtained with Spectral Sequencing and the Successive Augmentation heuristics are worse. On the other hand, in the random geometric graph, the best solutions are, in the average, obtained with Spectral Sequencing. However, independent runs of Simulated Annealing, hit a better solution. In both graphs, the running time of Simulated Annealing is much greater than the running time of Spectral Sequencing.

Comparing the differences between the percentage above the best layout ever seen, and assuming that the best layout ever seen is close to the optimum, it can be remarked that approximating geometric random graphs is harder than approximating uniform random graphs. In this sense, it looks like that almost any layout will not be very far from the optimum in $\mathcal{G}_{n,p}$ graphs. The results on the `randomA1` and `randomA2` graphs in the Appendix give strength to this conjecture.

6.6 Evaluation of the heuristics

Sumarized results to evaluate the performance of the heuristics on the rest of graphs are given in Appendix A. Here follow some observations that are worth to discuss.

It can observed that Spectral Sequencing is a method that, in general, produces good results in short time. Due to this fact, and because this is not a randomized algorithm, it seems a natural "standard" algorithm to compare against. This is the reason we have decided to normalize our results with Spectral Sequencing when the optimal cost is unknown.

From the boxplots, one can also see that the only method that always delivers better solutions than Spectral Sequencing is Simulated Annealing. However, Simulated Annealing is much more slower: For instance, Spectral Sequencing runs in one second on the `airfoil1` graph, but Simulated Annealing takes two hours.

Our observations also establish different trends existing between the approximation of the different families of graphs. From our measures, the Hillclimbing heuristic works well on binomial random graphs (even better than Spectral Sequencing), whereas it performs worse in graphs with an implicit geometric structure, such as random geometric graphs or finite elements computations graphs. In our set of FE graphs, Hillclimbing performs worst than successive augmentation, whereas the contrary happens for our set of VLSI instances.

Regarding the different neighboorhoods in Hillclimbing, we can observe on all graphs that the one that behaves better is Flip2, followed closely by Flip3, which is far away from FlipE. The initial ordering for the Greedy heuristic does not seem to follow any pattern on the considered graphs, although one can observe that the better orderings use to be either the random breadth search or the breadth search.

We would like to remark that there are great differences between the normalized costs: For some graphs these differences range from 1 to 80, while for others they range from 0.7 to 1.2. In the graphs where this range is small, one can may be

infer that the best and worst cost are not that far away (this is indeed the case for the hypercube and for the random binomial graphs, see [Díaz et al. 2001]). In the graphs where the range is large, no much conclusions can be drawn.

Finally (and surprisingly), for some "real life" graphs the normal heuristic provides quite good results! This can be due to the fact that data locality is often implicit in the vertex numbering or that the layout of these graphs has previously been optimized somehow to reduce their profile in order to improve the efficiency of some storage schemes.

6.7 The geometry of the solutions

In order to get a feeling of the characteristics of the layouts computed by each heuristic, we have devised a way to "view" them on the FE graphs, for which we have graphical information. For each heuristic, we have conveyed to the edges of this graph a color according to their cost $|\varphi(u) - \varphi(v)|$. Here, "cold colors" represent low costs and "hot colors" represent high costs. For the `airfoil1` graph, Figure 11(a) reproduces the Spectral solution, and Figure 11(b) reproduces a Simulated Annealing solution. A shadowing process has also been applied to this figure.

It is interesting to see in this way the quality and the geometry of the different heuristics considered. For instance, for the `airfoil1` graph, in Figure 11 one can appreciate that the solution obtained by Spectral Sequencing is quite uniform over all the edges, whereas the solution obtained by Simulated Annealing is made of different zones of lower cost separated by grooves with a higher cost.

Similar representations for `airfoil1` graph and other heuritics can be found at http://www.lsi.upc.es/∼jpetit/MinLA/Experiments/GraphicalView.

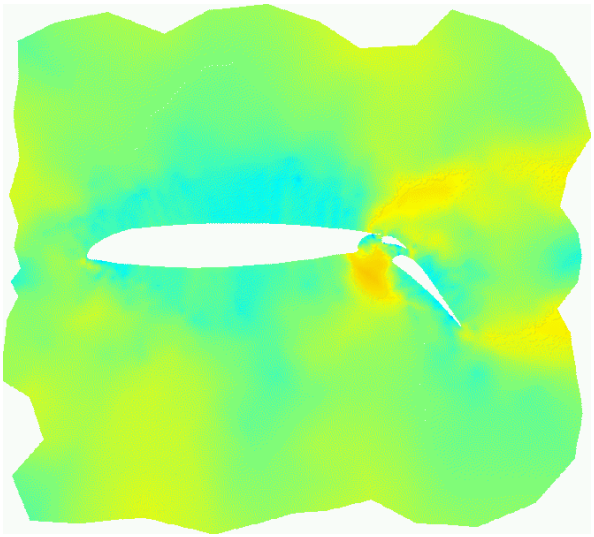
7. CONCLUSIONS

In this paper we have presented different heuristics to approximate the MinLA problem and to obtain lower bounds. These methods were applied to a testsuite made of a variety of sparse graphs (both regular, random and from "real life" applications) empirically obtaining the first comparative results on the MinLA problem. The case of dense graphs has not been considered, as fully polynomial time approximation schemes already exist.

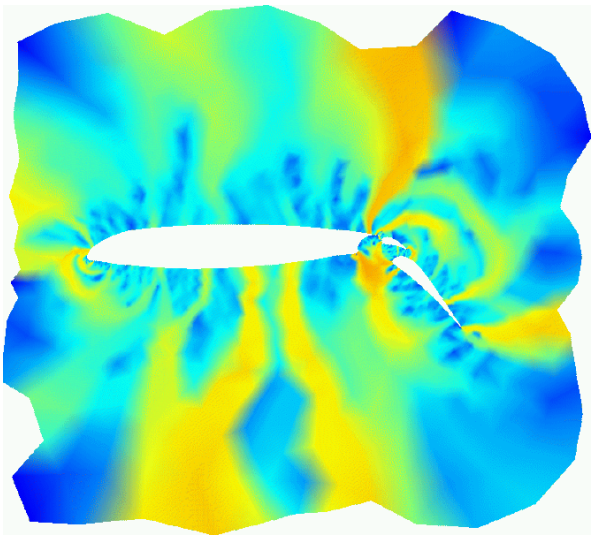
Several new techniques to find lower bounds have been presented, and we have observed that for certain classes of graphs, they can deliver better bounds than previous existing methods.

Most approximation heuristics are based on two well known general techniques, Auccessive Augmentation and Local Search. Adapting these general techniques to the particular problem of MinLA is easy, but decisions that have great effect on performance have to be made (such as the initial ordering of vertices in the successive augmentation heuristics, the neighborhood structure in local search, and parameter tuning for Simulated Annealing). Due to the lack of theoretical results in the literature, the only way to characterize these decisions is empirical testing. We have also experimented with Spectral Sequencing, which is related to a particular eigenvector of the Laplacian matrix of the graph.

The extensive experimentation and the benchmarking we have presented suggests that the best heuristic to approximate the MinLA problem on sparse graphs is Simulated Annealing when measuring the quality of the solutions. However, this



(a) Spectral Sequencing



(b) Simulated Annealing

(c) Scale

Fig. 11. Graphical view of two layouts on the `airfoil1` graph.

heuristic is extremely slow whereas Spectral Sequencing gives results not too far from those obtained by Simulated Annealing, but in much less time. In the case that a good approximation suffices, Spectral Sequencing would clearly be the method of choice. The price to pay in order to have even better approximations is the long execution time of Simulated Annealing. To a lesser extent, simpler methods, as Hillclimbing and Successive Augmentation with depth-first search can also obtain solutions close to the ones found by Spectral Sequencing.

Our study also establishes the big differences existing between the approximation of randomly generated graphs and graphs arising from real applications. From our experiments, the hillclimbing heuristic works well on general random graphs (even better than Spectral Sequencing), whereas it performs worse in graphs with an implicit geometric structure (such as random geometric graphs or finite elements computations graphs). In our set of FE meshes, hillclimbing performs worse than successive augmentation, whereas the contrary happens for our set of VLSI instances.

Experiments show that there exists a large gap between the best costs of the solutions obtained with several approximation heuristics and the best known lower bounds. The conclusion we draw is that in the practical sense, it is not only hard to get good upper bounds the MinLA problem, but that it is also difficult to get good estimates for the lower bounds.

ADDENDA

Since the presentation of a preliminar version of this paper in ALEX '98, the author, together with various reseachers, has devoted attention to several layout problems including MinLA in an analytical way. That theoretical study has been guided by the observations of this experimental research. For instance, in [Díaz et al. 2001] it has been shown that, with overwhelming probability, the cost of any arbitrary layout of a random uniform graph is within a small constant of the optimal cost, as suggested by the experiments on the `randomA*` graphs. On the other hand, the results in [Díaz et al. 2001; Díaz et al. 2000] are motivated by the observations made during this research on the FE graphs and the random geometric graphs.

On the other hand, the author has just launched the MinLA Repository, a client/server Internet system dedicated to this problem. This advanced repository contains four kinds of items: documents, graphs, solutions and programs. Everbody has read access to all these items and everybody is welcome to send new items to the administrator. Moreover, registered users can execute the available heuristics in this site on the available graphs. A Condor queue [Condor] handles the requests on a cluster of a dozen of PCs and the system keeps track of the best solutions found. Figure 12 depicts the system, which is available at http://tracer.lsi.upc.es/minla.

Finally, in [Petit 2003], the author presents and analyzes parallel heuristics to approximate MinLA. The heuristics consist in obtaining a first global solution using Spectral Sequencing and improving it locally through Simulated Annealing. These parallel heuristics will be soon included in the Repository.

ACKNOWLEDGEMENTS

The work done by M. Lobato Pagès (MarcLP) and T.J. Treangen on the MinLA repository is deeply acknowledged.

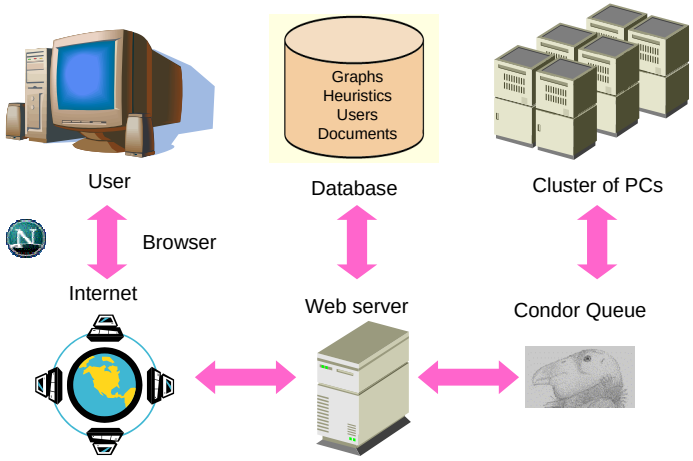


Fig. 12. Architecture of the MinLA Repository

REFERENCES

Adolphson, D. 1977. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing 6*, 40–54.

Adolphson, D. and Hu, T. C. 1973. Optimal linear ordering. *SIAM Journal on Applied Mathematics 25*, 3 (Nov.), 403–423.

Atkins, J. E., Boman, E. G., and Hendrickson, B. 1999. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing 28*, 1, 297–310 (electronic).

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M. 1999. *Complexity and approximation*. Springer-Verlag, Berlin.

Berry, J. W. and Goldberg, M. K. 1999. Path optimization for graph partitioning problems. *Discrete Applied Mathematics 90*, 27–50.

Burkard, R. E., Karisch, S. E., and Rendl, F. 1997. QAPLIB — A quadratic assignment problem library. *Journal of Global Optimization 391*, 10, 391–403.

Chung, F. R. K. 1988. Labelings of graphs. In *Selected topics in graph theory, 3*, pp. 151–168. San Diego, California: Academic Press.

Condor. The Condor project homepage. http://www.cs.wisc.edu/condor/.

Díaz, J., Penrose, M. D., Petit, J., and Serna, M. 2000. Convergence theorems for some layout measures on random lattice and random geometric graphs. *Combinatorics, Probability and Computing 9*, 6, 489–511.

Díaz, J., Penrose, M. D., Petit, J., and Serna, M. 2001. Approximating layout problems on random geometric graphs. *Journal of Algorithms 39*, 1, 78–116.

Díaz, J., Petit, J., and Serna, M. 2002. A survey on graph layout problems. Technical Report 3.

Díaz, J., Petit, J., Serna, M., and Trevisan, L. 2001. Approximating layout problems on random graphs. *Discrete Mathematics 235*, 1–3, 245–253.

Even, G., Naor, J., Rao, S., and Schieber, B. 1995. Divide-and-conquer approximation algorithms via spreading metrics. In *36th IEEE Symposium on Foundations of Computer Science* (1995), pp. 62–71.

Frieze, A. and Kannan, R. 1996. The regularity lemma and approximation schemes for dense problems. In *37th IEEE Symposium on Foundations of Computer Science*, pp. 12–20.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability. A guide to the theory of NP-completeness*. Freeman and Company.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. 1976. Some simplified NP-complete graph problems. *Theoretical Computer Science 1*, 237–267.

Goldberg, M. K. and Klipker, I. A. 1976. An algorithm for minimal numeration of tree vertices. *Sakharth. SSR Mecn. Akad. Moambe 81*, 3, 553–556. In Russian (English abstract at MathSciNet).

Gomory, R. E. and Hu, T. C. 1975. *Multi-terminal flows in a network*, pp. 172–199. Studies in Math., Vol. 11. Math. Assoc. Amer., Washington, D.C.

Harper, L. H. 1964. Optimal assignments of numbers to vertices. *Journal of SIAM 12*, 1 (March), 131–135.

Harper, L. H. 1970. Chassis layout and isoperimetric problems. Technical Report SPS 37–66, vol II (Sept.), Jet Propulsion Laboratory.

Hendrickson, B. and Leland, R. 1993. Multidimensional Spectral Load Balancing. *6th SIAM Conf. Parallel Proc. Sci. Comput.*

Hendrickson, B. and Leland, R. 1995. The Chaco user's guide.

Johnson, D. S. 1990. Local optimization and the traveling salesman problem. In *Automata, languages and programming (Coventry, 1990)*, pp. 446–461. New York: Springer.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Chevn, C. S. 1991. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research 39*, 3 (May), 378–405.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. 1989. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research 37*, 6 (Nov.), 865–892.

Juels, A. 1996. *Basics in Black-box Combinatorial Optimization*. Ph. D. thesis, University of California at Berkeley.

Juvan, M. and Mohar, B. 1992. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics 36*, 2, 153–168.

Kirkpatrik, S., Gelatt, C. D., and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science 220*, 671–680.

Klohs, K. 1999. parSA library user manual (version 2.1a).

Lang, K. and Rao, S. 1993. Finding near-optimal cuts: An empirical evaluation. In *Proceedings 4th Annual ACM-SIAM Symposium on Discrete Algorithms* (1993), pp. 212–221.

Liu, W. and Vannelli, A. 1995. Generating lower bounds for the linear arrangement problem. *Discrete Applied Mathematics 59*, 137–151.

Metropolis, W., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. 1953. Equation of State Calculations by Fast computing machines. *The Journal of Chemical Physics 21*, 6, 1087–1092.

Mitchison, G. and Durbin, R. 1986. Optimal numberings of an $n \times n$ array. *SIAM Journal on Algebraic and Discrete Methods 7*, 4, 571–582.

Muradyan, D. O. and Piliposjan, T. E. 1980. Minimal numberings of vertices of a rectangular lattice. *Akad. Nauk. Armjan. SRR 1*, 70, 21–27. In Russian (English abstract at MathSciNet).

Nakano, K. 1994. Linear layouts of generalized hypercubes. In *Graph-theoretic concepts in computer science (Utrecht, 1993)*, Volume 790 of *Lecture Notes in Computer Science*, pp. 364–375. Berlin: Springer.

Ousterhout, J. K. 1994. *Tcl and the Tk Toolkit*. Adisson–Wesley.

Pach, J., Shahrokhi, F., and Szegedy, M. 1996. Applications of the crossing number. *Algoritmica 16*, 111–117.

Papadimitrou, C. and Steiglitz, K. 1982. *Combinatorial Optimization, Algorithms and Complexity*. Prentice Hall.

Parlett, B., Simmon, H., and Stringer, L. 1982. Estimating the largest eigenvalue with the Lanczos algorithm. *Mathematics of Computation 38*, 157, 153–165.

Petit, J. 2003. Combining spectral sequencing and parallel simulated annealing for the MinLA problem. *Parallel Processing Letters 13*, 1, 77–91.

Rao, S. and Richa, A. W. 1998. New approximation techniques for some ordering problems. In *Proceedings 9th Annual ACM-SIAM Symposium on Discrete Algorithms* (1998), pp. 211–218.

Ravi, R., Agrawal, A., and Klein, P. 1991. Ordering problems approximated: single-processor scheduling and interval graph completition. In M. R. J. Leach, B. Monien Ed., *18th. International Colloquium on Automata, Languages and Programming*, Volume 510 of *Lecture Notes in Computer Science* (1991), pp. 751–762. Springer-Verlag.

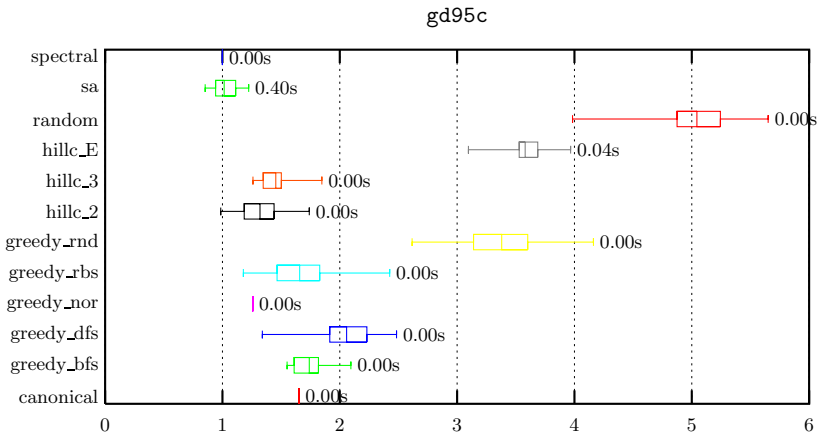
Reinelt, G. 1991. TSPLIB — A traveling salesman problem library. *ORSA Journal on Computing 3*, 376–384.

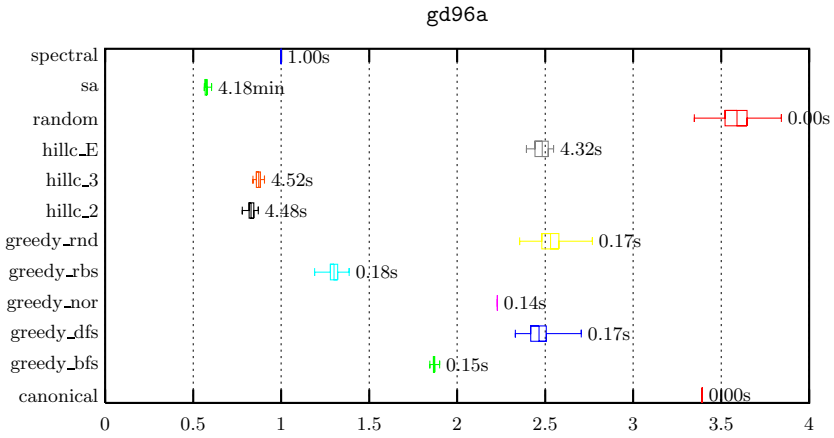
Shiloach, Y. 1979. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal on Computing 8*, 1 (Feb.), 15–32.

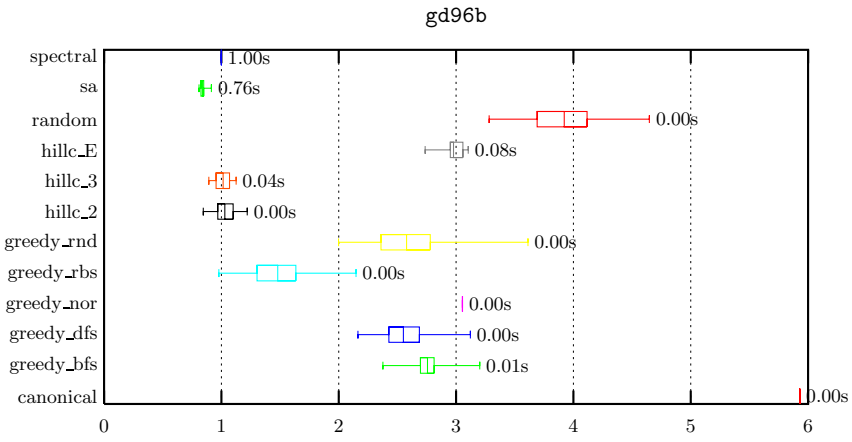
Skorobohatyj, G. 1994. Code for finding a minimum cut between all node pairs in an undirected graph. ftp://ftp.zib.de/pub/Packages/mathprog/mincut/all-pairs/index.html.

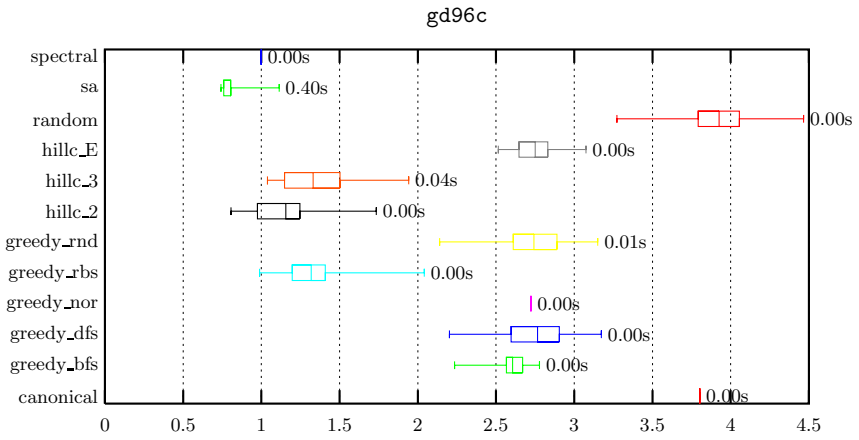
van Laarhoven, P. J. M. and Aarts, E. H. L. 1987. *Simulated annealing: theory and applications*, Volume 37 of *Mathematics and its Applications*. D. Reidel Publishing Co., Dordrecht.

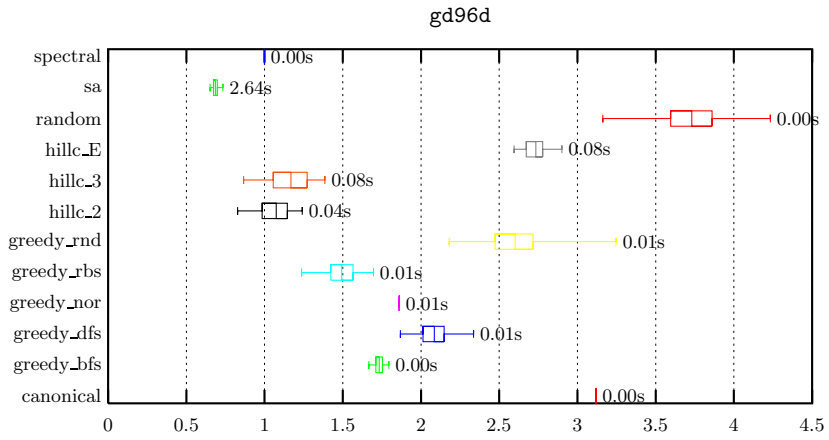
APPENDIX

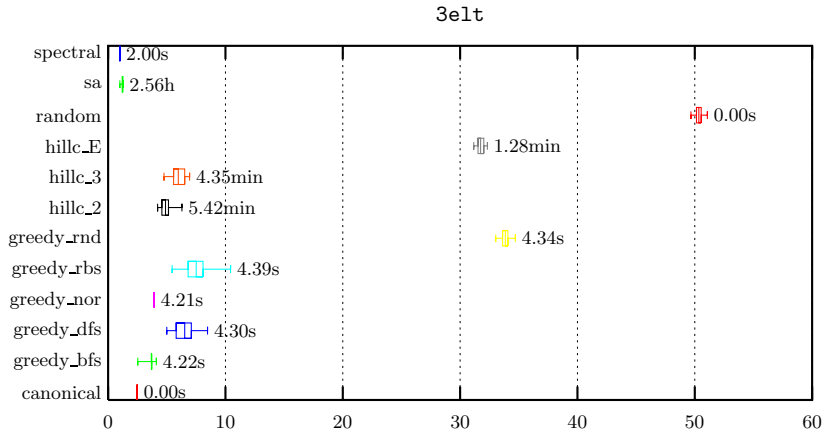


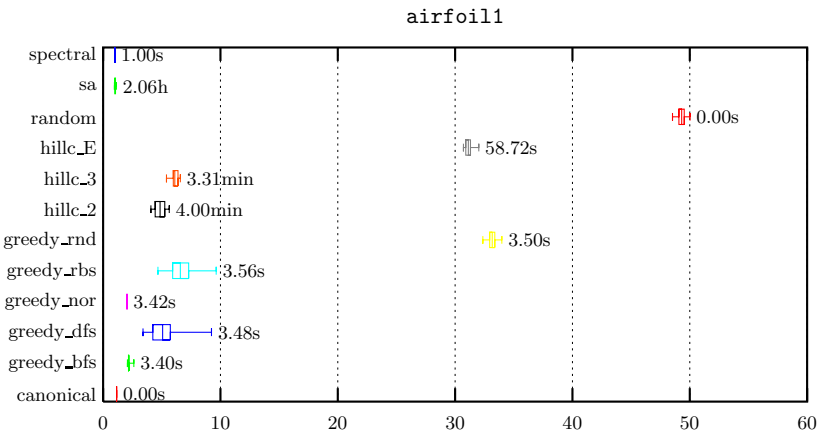


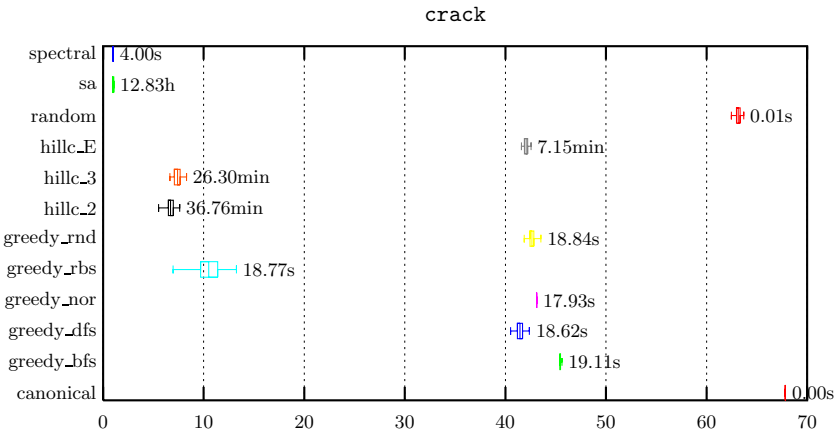


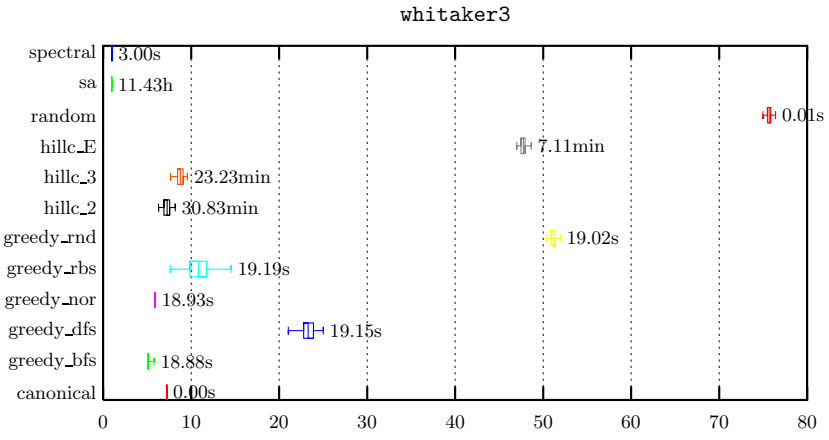

gd96c
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
0.00s
0.40s
0.00s
0.00s
0.04s
0.00s
0.01s
0.00s
0.00s
0.00s
0.00s
0.00s
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5

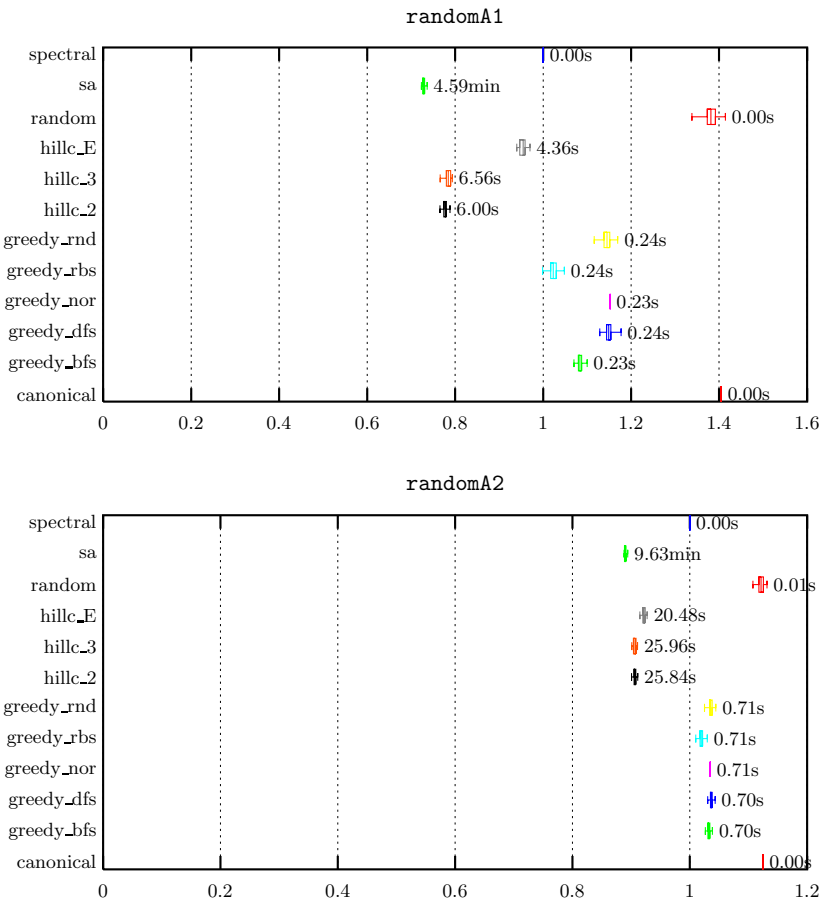

gd96d
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
0.00s
2.64s
0.00s
0.08s
0.08s
0.04s
0.01s
0.01s
0.01s
0.01s
0.00s
0.00s
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5


3elt
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
2.00s
2.56h
0.00s
1.28min
4.35min
5.42min
4.34s
4.39s
4.21s
4.30s
4.22s
0.00s
0 10 20 30 40 50 60


airfoil1
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
1.00s
2.06h
0.00s
58.72s
3.31min
4.00min
3.50s
3.56s
3.42s
3.48s
3.40s
0.00s
0
10
20
30
40
50
60


crack
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
4.00s
12.83h
0.01s
7.15min
26.30min
36.76min
18.84s
18.77s
17.93s
18.62s
19.11s
0.00s
0
10
20
30
40
50
60
70


whitaker3
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
3.00s
11.43h
0.01s
7.11min
23.23min
30.83min
19.02s
19.19s
18.93s
19.15s
18.88s
0.00s
0
10
20
30
40
50
60
70
80


randomA1
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
0.00s
4.59min
0.00s
4.36s
6.56s
6.00s
0.24s
0.24s
0.23s
0.24s
0.23s
0.00s
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
randomA2
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
canonical
0.00s
9.63min
0.01s
20.48s
25.96s
25.84s
0.71s
0.71s
0.71s
0.70s
0.70s
0.00s
0
0.2
0.4
0.6
0.8
1
1.2