Experiments on the Minimum Linear Arrangement Problem *

Jordi Petit

Abstract

This paper deals with the Minimum Linear Arrangement problem from an experimental point of view. Using a test-suite 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 test-suite 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. However, 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.

*This research was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and by the CICYT project TIC1999-0754-C03.

1 Introduction

Given an undirected graph $G = (V, E)$ with $|V| = n$, a *layout* 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

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

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 appears as problem number GT42 in [11] and as GT40 in [4]. 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 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 (MinLA) in order to minimize the total wire length [16, 2].

MinLA is also connected with 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.* [32] 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 [28] and has also been shown to be relevant to single machine job scheduling [1, 36].

The MinLA problem is known to be **NP**-hard and its decision version is **NP**-complete [12]. 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 [11].

The lack of efficient exact algorithms for general graphs has given rise to the possibility of finding approximation algorithms. In the case that the input graph is dense (that is, $|E| = \Theta(|V|^2)$), an approximation of MinLA within a $1 + \epsilon$ factor can be computed in time $n^{O(1/\epsilon)}$ for any $\epsilon > 0$ [10]. The first non-trivial approximation algorithm for MinLA on general graphs was presented in [9] and provides approximated solutions within a $O(\log n \log \log n)$ factor. To date, the best approximation algorithm gives an $O(\log n)$ approximation factor for general graphs [35]. However, that algorithm presents the disadvantage of having to solve

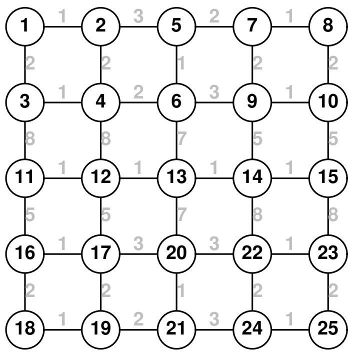


Figure 1: An optimal linear arrangement for the 5×5 mesh. The black numbers in the nodes show the labeling. The grey numbers represent the incurred costs for each edge, whose sum is 117.

Table 1: Survey of classes of graphs optimaly solvable in polynomial time.

Class of graph	Complexity	Ref.
Trees	$O(n^3)$	[13]
Rooted trees	$O(n \log n)$	[2]
Trees	$O(n^{2.2})$	[38]
Trees	$O(n^{\log 3/\log 2})$	[5]
Rectangular meshes	$O(n)$	[29]
Square meshes	$O(n)$	[28]
Hypercubes	$O(n)$	[15]
de Bruijn graph of order 4	$O(n)$	[16]
d-dimensional c-ary cliques	$O(n)$	[30]

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 [23, 26].

The goal of this paper is to analize different upper and lower bounding techniques for the MinLA problem. This is done running computational experiments on a selected test-suite of graphs. We focus on sparse graphs, for which no tight theoretical results exist. It is interesting to remark that the reduction used for prooving the **NP**-completeness of this problem creates dense graphs [12], 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 offers an interpreter from which it is possible to use the upper and lower bounding algorithms previously described. The architecture of this layout might be of interest to practitioners. Section 5 introduces the test-suite of graphs on which the experiments will be performed. This test-suite 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 analyze several computational experiments aimed to anlize 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.

The Path method [23]. Let $P_n^k = (V_n, E_n^k)$ denote the k-th power graph of the path P_n, where $V_n = \{1, \ldots, n\}$ and $E_n^k = \{ij \mid 0 < |i-j| \leq k\}$. It can be seen that

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

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

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

Juvan and Mohar prove in [23] the following theorem:

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

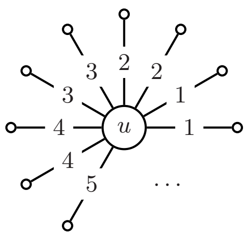


Figure 2: Illustration for the Degree method.

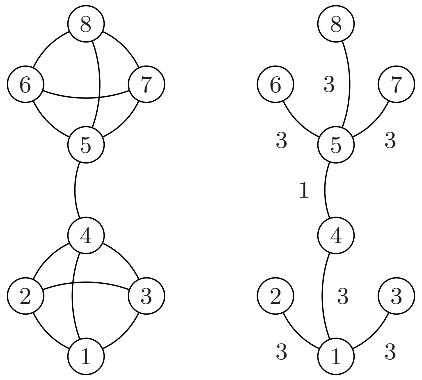
The Edges method [Petit]. Due to the floor operation taken in the previous theorem, the Path method ignores the length of some edges in the graph. A way to take them into account is to use the Edges method: Consider any layout φ. Notice that no more than $n-1$ edges can have cost 1 in φ. Moreover, no 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) : integer **is**
> $n := |V(G)|;\ m := |E(G)|;\ i := 1;\ f := 0;\ lb := 0$
>   **while** $f + n - i \leq m$ **do**
> $f := f + n - i$
> $lb := lb + i(n-i)$
> $i := i + 1$
>   **end while**
>   **return** $lb + i(m-f)$
> **end**

The Degree method [Petit]. 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).

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

The Gomory–Hu Tree method [2]. 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 [14]. The maximum flow $f[i, j]$



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	

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

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 by Adolphson and Hu [2] 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, $\text{LA}(G) \geq \sum_{e \in E'} w(e)$.

The Juvan–Mohar method [23]. Let $G = (V, E)$ be a connected graph with $V_n = \{1, \ldots, n\}$ 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 by Juvan and Mohar in [23]:

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

The Mesh method [Petit]. 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 [29] (and latter Mitchison and Durbin [28]) 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 in 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) : integer **is**
    $lb := 0$;
    Let M be the largest square mesh contained in G; let s be its side
    while $s \geq 2$ **do**
        $lb := lb + (4 - \sqrt{2})s^3$
        $G := G \setminus M$
        Let M be the largest square mesh contained in G; let s be its side
    end while
    return $lb + Edgesmethod(G)$
end

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).

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.

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.

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 [19, 20].

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) : integer is
    φ[v_i] := x;  c := 0
    for j := 1 to i do if v_i v_j ∈ E then c := c + |φ[v_i] − φ[v_j]|
    return c
end

function SuccessiveAugmentation(G) : φ is
    n := |V(G)|
    Select an initial ordering of vertices v_1, v_2, ..., v_n
    φ[v_1] := 0;  l := −1;  r := 1
    for i := 2 to n do
        if Increment(G, φ, i, v_i, l) < Increment(G, φ, i, v_i, r) then
            φ[v_i] := l;  l := l − 1       (∗ Put at left ∗)
        else
            φ[v_i] := r;  r := r + 1       (∗ Put at right ∗)
        end if
    end for
    (∗ Remap φ to {1, ..., n} ∗)
    for i := 1 to n do φ[i] := φ[i] − l
    return φ
end
```

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 and 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, 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.

Spectral Sequencing [23]. We review now a heuristic to find layouts due to Juvan and Mohar. 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

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

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 [3, 17].

Local Search heuristics. Local Search has been described by Papadimitriou and Steiglitz as "an area where intuition and empirical tests play a crucial role and where the design of effective Local Search is much an art" [33]. 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 worse 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 ($f : S \to \mathbb{R}^+$) and a neighborhood, which is a relation between feasible solutions that are "close" in some sense. The generic algorithm (subject to many variations) is as follows:

```
function LocalSearch is
    σ := Select initial random feasible solution
    while ¬Termination() do
        σ' := Select a neighbor of σ
        δ := f(σ) − f(σ')
        if Acceptable(δ) then σ := σ'
    end while
    return ⟨σ, f(φ)⟩
end
```

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 $\textsc{la}(G, \varphi)$. 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.

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 suitabl, 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.

Below, we present some black-box heuristics, derived from the general local search heuristic. These variations depend on the way the neighborhood is explored to search favorable moves or which is the criterion to accept moves that do not directly improve the solution. These algorithms are usually called *black-box heuristics* [22] because they work only with the objective function and the neighborhood structure but do not use problem-dependent strategies.

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, φ, 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*(φ, u, v) performs this flipping.

```
function HillClimbing2(G, max) is
    φ := Generate an initial random layout
    z := 0
    repeat
        z := z + 1
        u := Generate a random integer in {1, ..., n}
        v := Generate a random integer in {1, ..., n}
        δ := GainWhenFlip2(G, φ, u, v)
        if δ ≥ 0 then
            Flip2(φ, u, v)
            if δ > 0 then z := 0
        end if
    until z = max
    return ⟨φ, C_{G,φ}⟩
end
```

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=$\log_2^2 n$.

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, max) is
    φ := Generate an initial random layout
    z := 0
    repeat
        z := z + 1
        ⟨u, v, δ⟩ := SelectBestMove(G, φ)
        if δ ≥ 0 then
            Flip2(φ, u, v)
            if δ > 0 then z := 0
        end if
    until z = max
    return ⟨φ, C_{G,φ}⟩
end
```

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.

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 [27] is parametrized by a temperature t and proceeds as follows (using the Flip2 neighborhood):

```
function Metropolis(G, r) is
    φ := Generate an initial random layout
    for i := 1 to r do
        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}) do Flip2(φ, u, v)
    end for
    return ⟨φ, C_{G,φ}⟩
end
```

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.

Simulated Annealing. The Simulated Annealing heuristic [24] 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) **is**
    $\varphi :=$ Generate an initial random layout
    $t :=$ *InitialTemperature()*
    while $\neg$*Frozen()* **do**
        while $\neg$*Equilibrium()* **do**
            $u :=$ 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 $\min\left(1, e^{-\delta/t}\right)$ **do** *Flip2*(φ, u, v)
        end while
        $t := \alpha \cdot t \qquad (* \; 0 < \alpha < 1 \; *)$
    end while
    return $\langle \varphi, C_{G,\varphi} \rangle$
end

The main point in SA algorithms is the selection of their parameters (initial temperature, frozing and equilibrium detection, cooling ratio α …). 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 [19, 21, 20]. Rather than investigating different selections for these parameters, we have used an adaptative technique due to Aarts.

4 The `llsh` 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 Minimum Linear Arrangement problem, which we briefly describe in this section.

The architecture of this toolkit contains two layers. In the inner layers reside 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 enables using the core using a Ousterhout:1994:TTT interpreter [31]. 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 coding of the driver programs that perform the experiments and process the results.

The development of the core of the toolkit has devoted mu attention to provide efficient implementations. To achieve this goal without much cost in 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 [34] offered in the Chaco library [18].
- In order to compute the Gomory–Hu tree of a graph, a public domain implementation by G. Skorobohatyj has been ported to C++ [39].
- The simulated annealing heuristic has been completely implemented using the parSA library [25], which offers the Aarts adaptative scheduler.

In order to illustrate how the interpreter of the toolkit works, Figure 4 shows a sample session.

`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`

Figure 4: Sample session with `llsh`.

5 A test suite for evaluating MinLA methods

In contrast to some other combinatorial optimization problems for which complete libraries of instances exist (consider for example the TSPLIB for the Traveling Salesperson problem [37], no real large instances are publicly available for the MinLA problem. Since our aim is to measure the heuristics on sparse graphs, the graphs we have selected for our test-suite belonging to the following families:

- Uniform random graphs $\mathcal{G}_{n,p}$: Graphs with n nodes and probability p to have an edge between any pair of nodes. As we are interested on sparse graphs, p will be chosen small, but large enough to ensure connected graphs.
- Geometric random graphs $\mathcal{G}_n(d)$: Graphs with n nodes located randomly in a unit square. Two nodes will be connected by an edge if the (Euclidean) distance between them is at most d.
- Graphs with known optima: trees, hypercubes, meshes.
- Graphs from finite element discretizations (FE class).
- Graphs from VLSI design (VLSI class).
- Graphs from graph-drawing competitions (GD class).

Except for the graph-drawing competitions graphs, all the graphs included in the test-suite have 1000 or more vertices, which represent big and challenging instances for current heuristics. The main characteristics of our test-suite are shown on Table 2.

Graphs `randomA*` are uniform random graphs with different edge probabilities. The graph `randomA4` was designed to have a similar number of nodes and edges than `randomG4`, in order to discover differences between uniform random graphs and random geometric graphs.

6 Experimental evaluation

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

Table 2: Test-suite.

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

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	$\mathcal{G}_{n=1000,p=0.05}$
randomA3	1000	49820	72/99.64/129	4	$\mathcal{G}_{n=1000,p=0.1}$
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$\times$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	

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 (expect 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 generators (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). Most results will be summarized using boxplots. These boxplots show the values of the minimum, first quartile, median, last quartile and maximum of a sample. The average time elapsed to compute an individual of the sample is displayed at the right of the corresponding boxplot. Table 3 resumes the abbreviations shown in other figures or tables.

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 at

http://www.lsi.upc.es/~jpetit/MinLA/Experiments/

on the World Wide Web.

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 reserach, are shown in Table 4.

The first fact that can be observed in Table 4 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 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

Table 3: Abreviations used in figures and tables.

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	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

poor performance of the Gomory–Hu tree method is that, for all the graphs in our test-suite, 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 "never finishes" on the random graphs.

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

Comparing the Flip2 and Flip3 neighborhoods. Notice that when using full exploration, the Flip3 neighborhood will be get stucked at less 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

Table 4: Comparison of the lower bounds methods.

Graph	Edges	Degree	Path	J–M	G–H	Mesh	best
`randomA1`	14890	16176	9970	140634	9926	†	867570
`randomA2`	321113	323568	319475	4429294	49404	†	6528780
`randomA3`	1288066	1277077	1280474	11463259	99511	†	14202700
`randomA4`	37713	39531	35796	601130	16325	†	1721670
`randomG4`	37677	39972	35796	14667	16315	†	169128
`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	217220
`airfoil1`	24112	24220	12754	5571	24569	40221	297222
`crack`	60424	64938	30715	25826	60751	95347	1491126
`whitaker3`	57571	57824	29395	11611	57970	144854	1151064
`c1y`	2767	14101	2479	13437	3192	2819	62936
`c2y`	3370	16473	2935	17842	3877	3762	80134
`c3y`	4555	20874	3976	23417	5320	5548	124712
`c4y`	4651	16404	4093	21140	5518	5778	116382
`c5y`	4069	16935	3601	19217	4790	4626	100264
`gd95c`	250	292	181	36	255	174	395
`gd96a`	2257	4552	1095	5155	3233	377	96342
`gd96b`	276	702	110	43	305	113	1470
`gd96c`	186	191	64	38	241	130	524
`gd96d`	277	595	179	415	331	113	2417

† The Mesh method could not be applied
* Optimal

average result of performing this experiment 100 times is shown in Figure 5.

As the Figure 5 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.

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 6, which shows the performance of the heuristics both in approximation quality and running time. For the sake of clarity, absolute values have been normalized by the optimal values, given in Table 4.

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 a 25% of the tries; for the mesh (`mesh33x33`), Simulated Annealing finds solutions not exceding a 44% of the optimum; for the binary tree (`bintree10`) the solutions obtained by Simulated

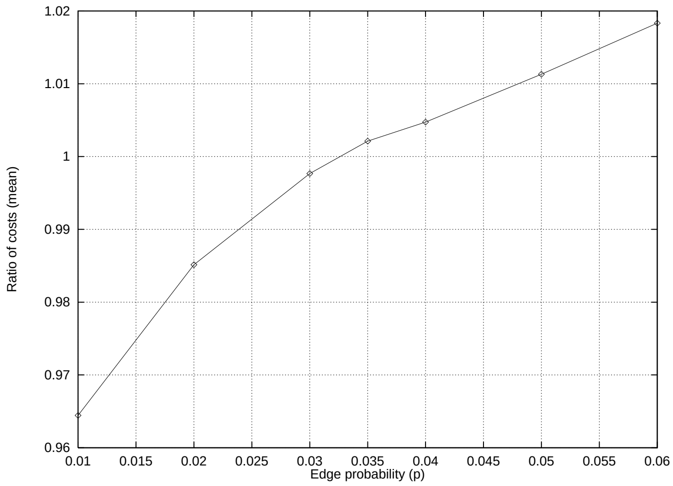


Figure 5: Comparison between the Flip2 and Flip3 neighborhoods in the Hill-climbing algorithm on $\mathcal{G}_{n=1000,p}$ random graphs. The curve shows the ratio between the average results obtained using Flip2 and Flip3 on 100 different runs.

Annealing almost double the optimal value.

Depending on the graph, some methods work better than others. In any case Simulated Annealing always dominates them, in spite 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.

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` 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 Figuree 7.

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 Hill-climbing 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`, `randomA2` and `randomA3` graphs in the Appendix give strength to this conjecture.

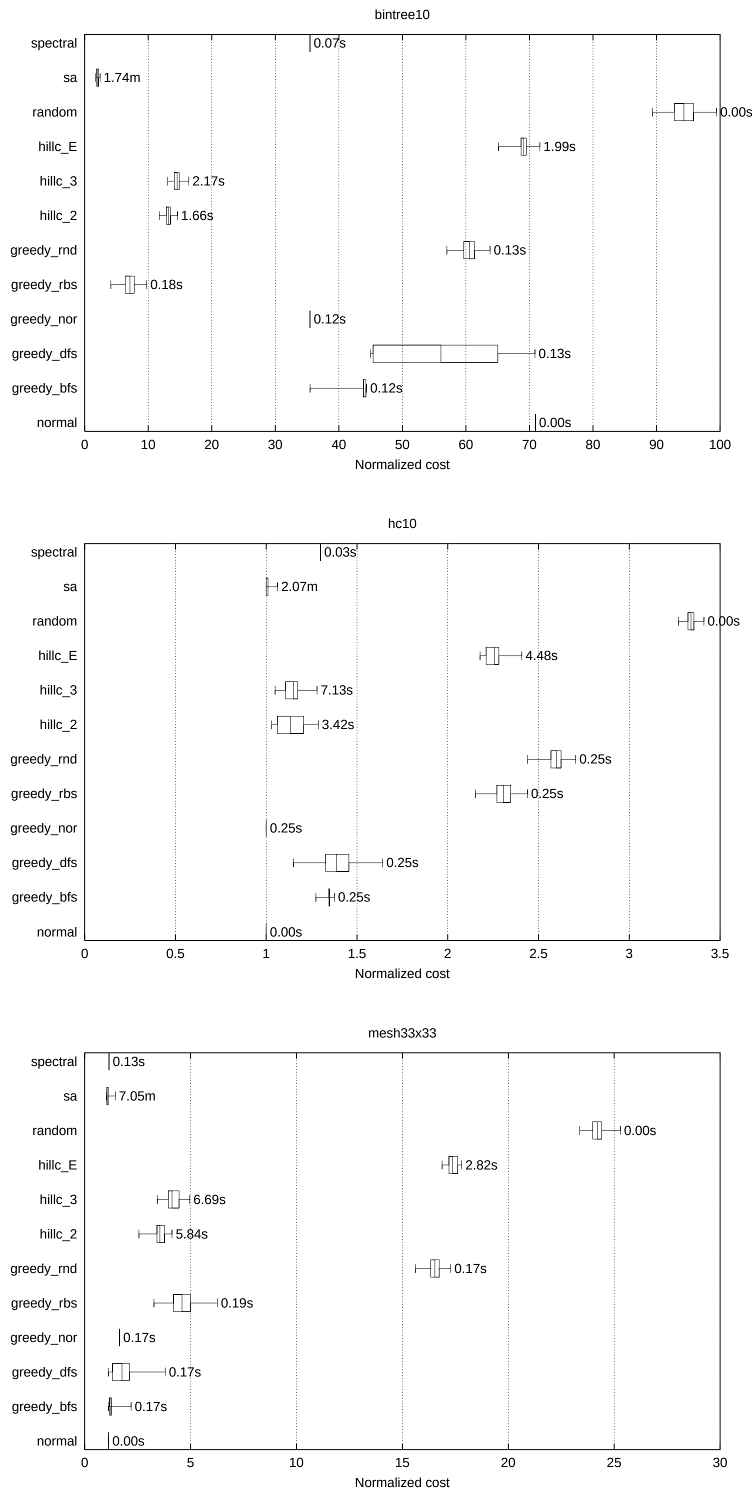


Figure 6: Comparison of heuristics for graphs with known optima. The absolute costs have been normalized dividing them by the respective optimal costs.

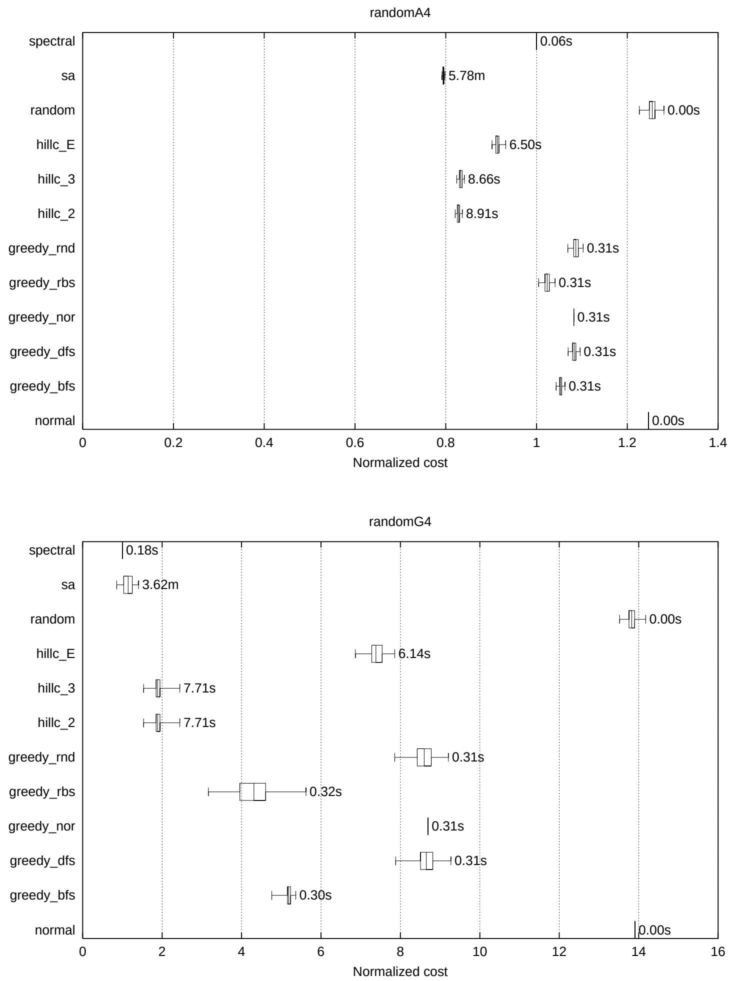


Figure 7: Comparison of heuristics for a uniform random graph and a random geometric graph with similar number of edges and nodes. The absolute costs have been normalized dividing them by the cost found by Spectral Sequencing.

Evaluation of the heuristics. Sumarized results to evaluate the performance of the different heuristic on the rest of graphs are given in Appendix A. There are some observations that are worth to discuss.

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 to reduce their profile in order to improve the efficiency of some storage schemes.

Aa for the graphs with know minima, it can be remarked that some methods work better than others depending on the class of graph, but that Simulated Annealing always dominates them, at the cost of having a longer running time.

The geometry of the solutions. In order to have a "graphical colored view" of the solutions delivered by each heuristic, please refer to

http://www.lsi.upc.es/~jpetit/MinLA/Experiments/GraphicalView/

on the World Wide Web, where we focus on the `airfoil1` graph, for which we have information on the coordinates of its nodes. For each heuristic, we have painted the edges of this graph according to their cost $|\varphi(u) - \varphi(v)|$: red edges have a cost greater than 200; orange edges have cost between 100 and 200; green edges have a cost between 50 and 100; and blue edges have at most cost 50. It is very interesting to see in this way the quality and the geometry of the different heuristics presented.

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 test-suite 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 many decisions that have a great effect on their behavior have to be taken (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 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 less 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 worst than successive augmentation, whereas the contrary happens for our set of VLSI instances.

Experiments evidence 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.

Colophon. 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 [8] 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 [6, 7] are motivated by the observations made during this research on the FE graphs and the random geometric graphs.

References

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

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

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

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

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

[6] J. Diaz, M. D. Penrose, J. Petit, and M. Serna. Approximating layout problems on random geometric graphs. *Journal of Algorithms.* To appear.

[7] J. Diaz, M. D. Penrose, J. Petit, and M. Serna. Convergence theorems for some layout measures on random lattice and random geometric graphs. To appear in Issue 4 of year 2000.

[8] J. Diaz, J. Petit, M. Serna, and L. Trevisan. Approximating layout problems on random graphs. *Discrete Mathematics.* To appear.

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

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

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

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

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

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

[15] L. H. Harper. Optimal assignments of numbers to vertices. *Journal of SIAM*, 12(1):131–135, Mar. 1964.

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

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

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

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

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

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

[22] A. Juels. *Basics in Black-box Combinatorial Optimization*. PhD thesis, University of California at Berkeley, 1996.

[23] M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992.

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

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

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

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

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

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

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

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

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

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

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

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

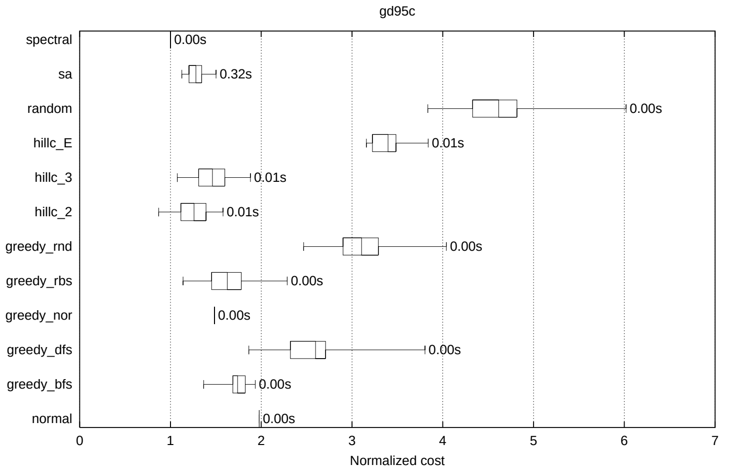
[36] R. Ravi, A. Agrawal, and P. Klein. Ordering problems approximated: single-processor scheduling and interval graph completition. In M. R. J. Leach, B. Monien, editor, *18th. International Colloquium on Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages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

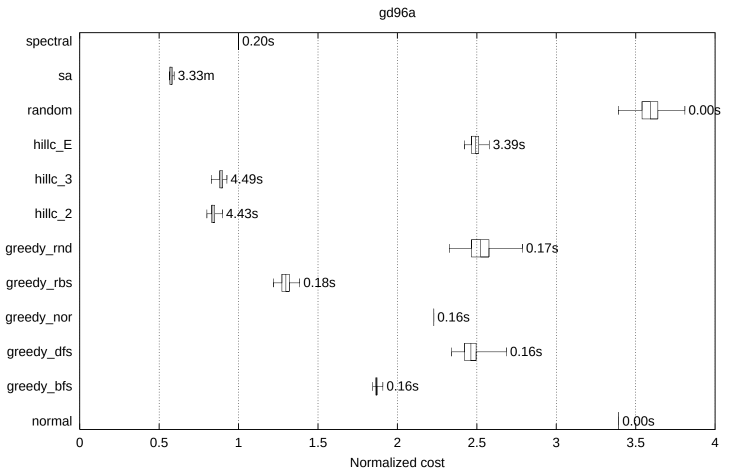
[37] G. Reinelt. TSPLIB, 1995.

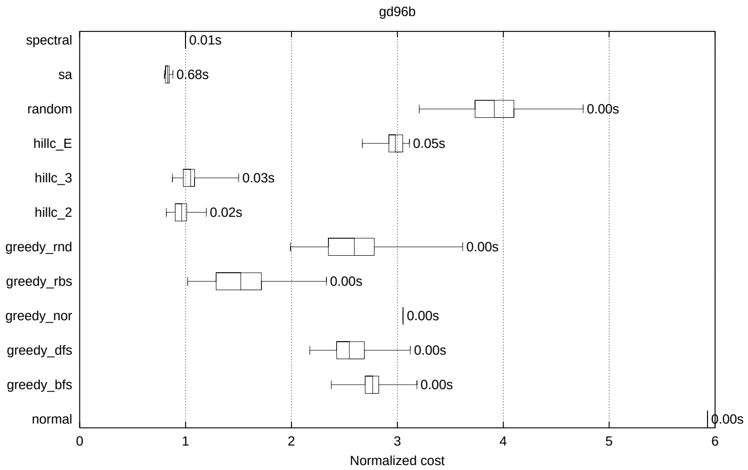
[38] Y. Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal on Computing*, 8(1):15–32, Feb. 1979.

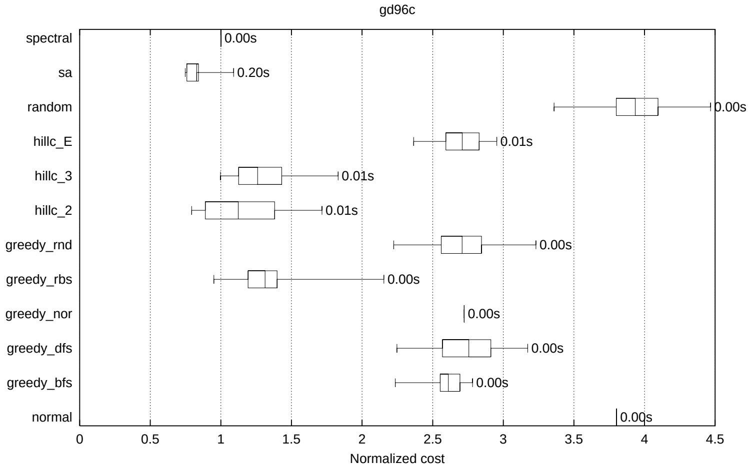
[39] G. Skorobohatyj. Code for finding a minimum cut between all node pairs in an undirected graph, 1994. ftp://ftp.zib.de/pub/Packages/mathprog/mincut/all-pairs/index.html.

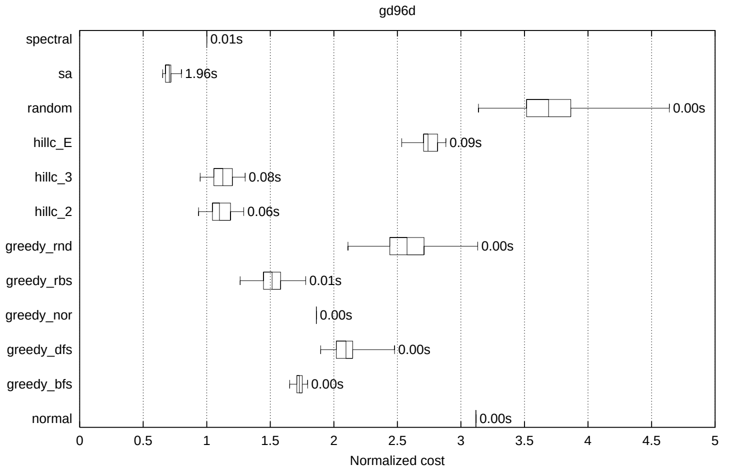
A Appendix

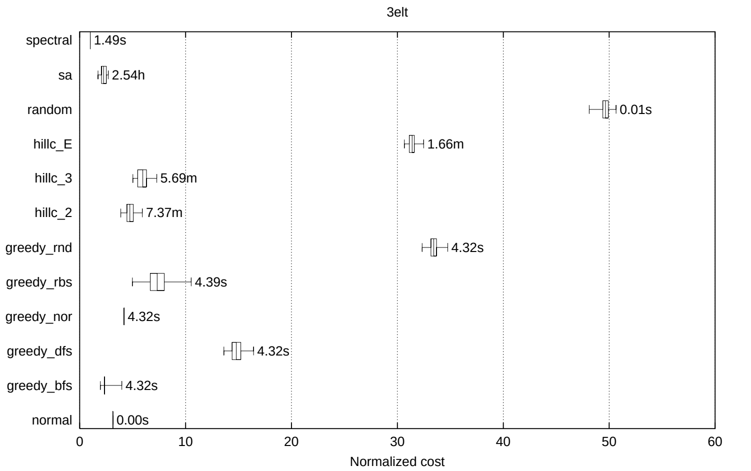

gd95c
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.00s
0.32s
0.00s
0.01s
0.01s
0.01s
0.00s
0.00s
0.00s
0.00s
0.00s
0.00s
0
1
2
3
4
5
6
7
Normalized cost

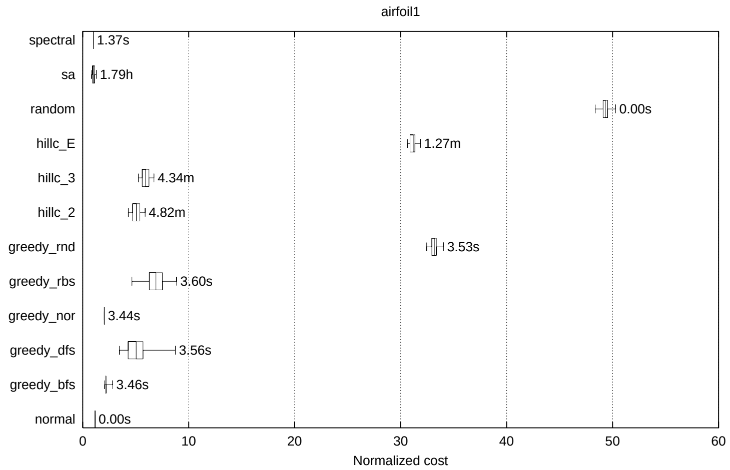

gd96a
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.20s
3.33m
0.00s
3.39s
4.49s
4.43s
0.17s
0.18s
0.16s
0.16s
0.16s
0.00s
0
0.5
1
1.5
2
2.5
3
3.5
4
Normalized cost

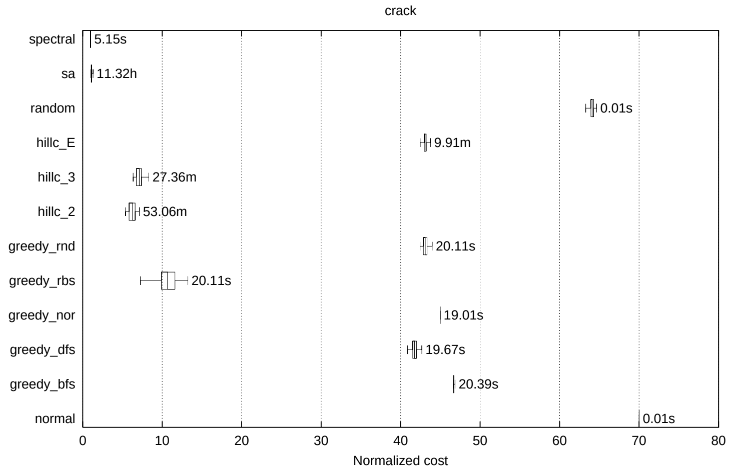

gd96b
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.01s
0.68s
0.00s
0.05s
0.03s
0.02s
0.00s
0.00s
0.00s
0.00s
0.00s
0.00s
0
1
2
3
4
5
6
Normalized cost

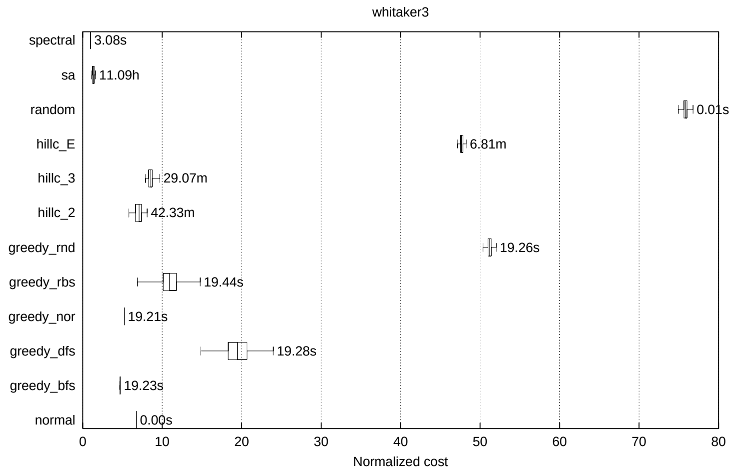

gd96c
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.00s
0.20s
0.00s
0.01s
0.01s
0.01s
0.00s
0.00s
0.00s
0.00s
0.00s
0.00s
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Normalized cost

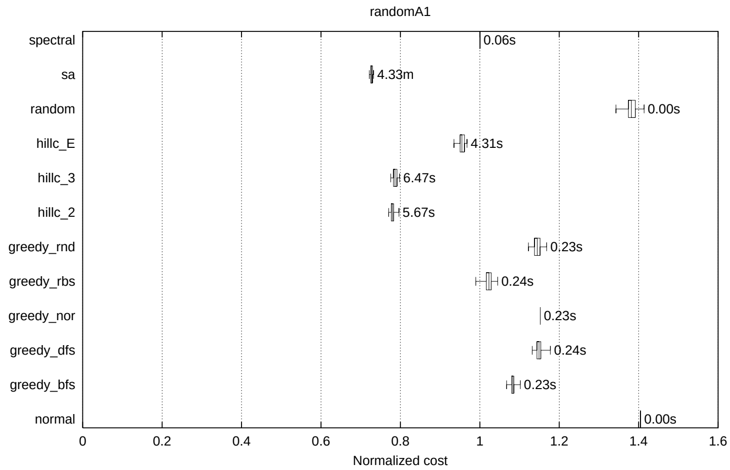

gd96d
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.01s
1.96s
0.00s
0.09s
0.08s
0.06s
0.00s
0.01s
0.00s
0.00s
0.00s
0.00s
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Normalized cost

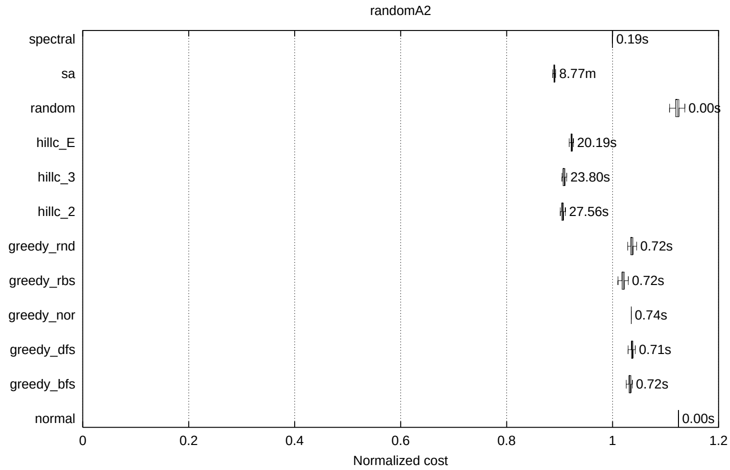

3elt
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
1.49s
2.54h
0.01s
1.66m
5.69m
7.37m
4.32s
4.39s
4.32s
4.32s
4.32s
0.00s
0
10
20
30
40
50
60
Normalized cost

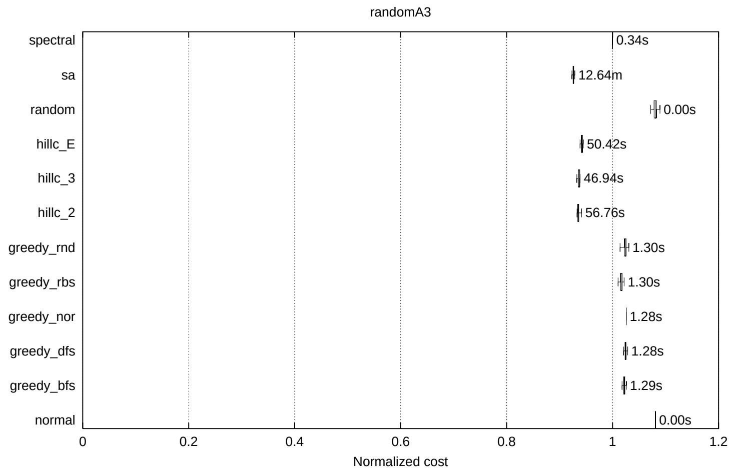

airfoil1
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
1.37s
1.79h
0.00s
1.27m
4.34m
4.82m
3.53s
3.60s
3.44s
3.56s
3.46s
0.00s
0
10
20
30
40
50
60
Normalized cost


crack
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
5.15s
11.32h
0.01s
9.91m
27.36m
53.06m
20.11s
20.11s
19.01s
19.67s
20.39s
0.01s
0
10
20
30
40
50
60
70
80
Normalized cost


whitaker3
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
3.08s
11.09h
0.01s
6.81m
29.07m
42.33m
19.26s
19.44s
19.21s
19.28s
19.23s
0.00s
0
10
20
30
40
50
60
70
80
Normalized cost


randomA1
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.06s
4.33m
0.00s
4.31s
6.47s
5.67s
0.23s
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
Normalized cost


randomA2
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.19s
8.77m
0.00s
20.19s
23.80s
27.56s
0.72s
0.72s
0.74s
0.71s
0.72s
0.00s
0
0.2
0.4
0.6
0.8
1
1.2
Normalized cost


randomA3
spectral
sa
random
hillc_E
hillc_3
hillc_2
greedy_rnd
greedy_rbs
greedy_nor
greedy_dfs
greedy_bfs
normal
0.34s
12.64m
0.00s
50.42s
46.94s
56.76s
1.30s
1.30s
1.28s
1.28s
1.29s
0.00s
0
0.2
0.4
0.6
0.8
1
1.2
Normalized cost