Heuristics for the MinLA Problem:
Some Theoretical and Empirical Considerations

Josep Diaz  
Jordi Petit i Silvestre  
Paul Spirakis

Heuristics for the MinLA Problem: Some Theoretical and Empirical Considerations*

Josep Díaz†   Jordi Petit i Silvestre†   Paul Spirakis†

Abstract

This paper deals with some aspects on finding good solutions for the Minimum Linear Arrangement problem (MinLA). We consider some random type of sparse graphs, for which it is possible to obtain trivial constant approximations. For similar graphs, we prove that Metropolis can find good solutions, whereas we exhibit an instance for which Hill Climbing is expected to need an exponential number of steps to hit an optimum. Motivated by the last results, we present an heuristic (SS+SA) to approximate the MinLA problem on sparse graphs. The heuristic consists in using Spectral Sequencing to obtain a first primal solution and after improving it locally through (Parallel) Simulated Annealing. In the last part of the paper, we report experimental results obtained with the SS+SA heuristic when applied to a set of large sparse geometric graphs. Compared to other heuristics, the measurements obtained on an IBM SP2 computer with 8 processors show that the new heuristic improves the solution quality, decreases the running time and offers an excellent speedup when ran in parallel.

1 Introduction

Given an undirected graph \( G = (V, E) \) with \( |V| = n \), a layout of \( G \) is a one-to-one function \( \varphi : V \to [1..n] \). The minimum linear arrangement problem (MinLA) is a combinatorial optimization problem formulated as follows: given a graph \( G = (V, E) \), find a layout \( \varphi \) of \( G \) that minimizes the cost

\[
c_\varphi = \sum_{u,v \in E} |\varphi(u) - \varphi(v)|.
\]

MinLA is an interesting and appealing problem that appears as problem number GT42 in [8] and as GT40 in [2], with several different applications in Computer Science and Biology [1, 9, 22]. The MinLA problem is known to be NP-hard and its decision version is NP-complete [8]. However, there exist exact solutions for some particular kinds of graphs in polynomial time [9, 22, 1, 27, 4, 24]. The lack of efficient exact algorithms for general graphs has given rise to the possibility of finding approximation algorithms. An approximation of MinLA within a \( 1 + \epsilon \) factor in time \( n^{O(1/\epsilon)} \) for dense graphs is presented in [7] and an \( O(\log n \log \log n) \) approximation for general graphs in [6]. An heuristic algorithm based on Spectral Sequencing was proposed in [16]. Techniques to find lower bounds are reported in [16, 18, 26].

*Supported by ESPRIT LTR Project no. 20244 — ALCOM-IT and CICYT Project TIC97-1475-CE.
†Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya. Campus Nord C6. Jordi Girona Salgado, 1-3. 08034 Barcelona, Spain. {diaz,jpetit}@lsi.upc.es Fax: +34 3 4017014
‡Computer Technology Institute (CTI). Patras, Greece. spirakis@cti.gr
In this paper we are interested in sparse graphs whose optimal layout is far from the average. This happens, in particular, for sparse graphs that are embedded in some geometric structure, as graphs arising from finite element discretizations, VLSI design or networks of computers. For this type of graphs, no particular tight approximation results are known. Moreover, it is an open problem to show if MinLA is still NP-complete on sparse graphs.

The second author has presented a number of heuristics to approximate MinLA and has measured their behavior on a set of benchmarking graphs [26]. The main conclusion was that for the type of graphs we are concerned with, the best results were obtained with Spectral Sequencing (SS) and Simulated Annealing (SA). It was found, however, that the time required by the SA process to improve the solution found by SS was inordinate.

The rest of this paper is organized as follows. Section 2 reviews some basic concepts in heuristics. In Section 3 we present some theoretical considerations on Hill Climbing and Metropolis for MinLA on particular classes of quasi-linear graphs. In Section 4, we address the problem of accelerating some of the heuristics in [26]. We do so by first presenting the sequential SS+SA heuristic and then two methods to parallelize it. Finally, in Section 5 we give experimental results to evaluate the proposed heuristic on sequential and parallel environments.

2 Basic Heuristics for the MinLA Problem

The MinLA problem can be seen as a black-box minimization problem, therefore Local Search could be a reasonable method to obtain approximations. The first Local Search algorithm one can use is Hill Climbing (HC). The basic principle of HC is to improve iteratively an initial randomly generated solution by performing local changes on its combinatorial structure. Only the changes that improve or keep the objective function are accepted. In general, HC does not assure that the found solution is also an optimal one. In [26] three different neighborhoods for HC on the MinLA problem were analyzed. It was observed that for sparse graphs the best results were obtained with the Flip2 neighborhood: 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. In this neighborhood it is easy to perform movements and to compute their gain.

One way to avoid being stuck in local optima is to accept with small probability downhill moves. Metropolis [21] proposed an heuristic parameterized by a temperature \( t \). A move that produces a gain of \( \delta \) is accepted with probability \( \min\{1, e^{-\delta/t}\} \). A refined heuristic is Simulated Annealing (SA) [17]. The SA algorithm 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). Since under certain conditions SA is ergodic (i.e., its stationary distribution does not depend on the initial solution), practitioners use to start it with a randomly generated solution. A key point to apply SA to a given problem is the selection of its parameters (such as the cooling schedule, freezing and equilibrium detection...) [15, 26].

A general framework to parallelize SA on distributed memory machines is presented in [5]. Some other papers have presented data level parallelizations for SA for other placement problems (e.g. [3]). In these approaches, the problem is split into subsets that are distributed among the available processors. Each processor performs sequential SA on its own subset of data. As there can be dependencies between moves performed in different processors, two methods have been proposed in order to reduce the need of frequent synchronizations: exact methods block some movements in order for different processors to evolve on the basis of a
coherent state; chaotic methods admit that the gain computation involves some errors (that are expected to be small).

Another heuristic particularly appropriated to the MINLA problem is Spectral Sequencing (SS) [16]. Let $G = (\{1..n\}, E)$ denote a connected graph and let $L_G$ stand for its Laplacian matrix with entries $L_G[u,v]$ equal to $\deg(u)$ if $u = v$, to $1$ if $u$ and $v$ are adjacent, and to $0$ otherwise. As $L_G$ is positive semidefinite, its smallest eigenvalue is zero. Let $\lambda_2$ be the second smallest eigenvalue of $L_G$ and $x^{(2)}$ its eigenvector. The vector $x^{(2)}$ is called the Fiedler vector of $G$. SS computes $x^{(2)}$ and after ranks each position of $x^{(2)}$. Thus, the heuristic returns an arrangement $\phi$ satisfying $\phi(u) \leq \phi(v)$ whenever $x^{(2)}_u \leq x^{(2)}_v$.

It is expected that SS produces good results because the ordering of 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 already been used in other problems such as graph partitioning, chromosome mapping or matrix reordering (see e.g. [28]). The computational intensive part of the SS heuristic is the computation of the Fiedler vector. It is usually implemented via the Lanczos algorithm [25, 10]. Spectral techniques also offer a lower bound for the MINLA problem:

**Theorem 1** ([16]) For any graph $G$ and any layout $\phi$ on $G$, $c_\phi \geq \lambda_2(n^2 - 1)/6$.

### 3 Quasi-Linear Graphs, Metropolis and Hill Climbing

In this section, we present three kinds of linear graphs (random linear graphs, line graphs and hidden lines) and analyze their approximation properties. All the sketches of the proofs can be found in Appendix A.

#### 3.1 Random linear graphs and normal layouts

Let $L_n$ be the class of random linear graphs on vertex set $V = \{1..n\}$ and where each possible edge $\{i,j\}$ appears in the graph with probability $1/(j-i+1)$. It is straightforward to see that the expected number of edges in a $L_n$ graph is $O(n \log n)$. The normal layout $\psi$ of a $L_n$ is defined by $\psi(i) = i$ ($\forall i \in \{1..n\}$). Random linear graphs are useful to generate MINLA instances for which a good estimate of the optimum is known by construction but is somehow hidden from the heuristics.

**Lemma 1** With high probability, $E[c_\phi] = n^2/2 + O(n \log n)$.

Since $\forall \phi : c_\phi \geq \lambda_2(n^2 - 1)/6$, we have asymptotically that $\lambda_2 \leq 3$. Therefore,

**Lemma 2** The normal layout of a random linear graph is a constant approximation to its MINLA value.

#### 3.2 Hill Climbing can fail on line graphs

Let $I_n$ be the line graph with vertex set $V = \{1..n\}$ and edge set $\{\{i,i+1\}\}_{i=1}^{n-1}$. This graph has only two layouts with minimal cost: $\phi_1^*(i) = i$ and $\phi_2^*(i) = n-i+1$ with $c(\phi_1^*) = c(\phi_2^*) = n-1$. Moreover, the maximum cost value for a layout in $I_n$ is $n^2/2 + n$ and the expected value of the cost of a randomly generated layout is $O(n^2)$.
In the following, we consider the application of Hill Climbing with the Flip2 neighborhood on a line graph. This algorithm on this class of graphs has a nice property:

**Lemma 3** From any starting layout \( \pi \), after a sufficiently long number of steps, the HC algorithm on a line graph always hits a minimal layout.

The question is how much time will HC spend in finding an optimum layout.

**Lemma 4** There is a layout \( \pi_s \) of \( I_n \) (\( \pi_s \) is characterized in Appendix A) such that HC needs at least \( \mathcal{O}(n^3) \) steps to hit an optimum.

Unfortunately, the probability of reaching an optimum starting from \( \pi_s \) in \( \mathcal{O}(n^3) \) steps is smaller than \( \mathcal{O}(n^{-n}) \). The reason is that HC "can get lost" in many plateaus. In fact:

**Theorem 2** Starting from \( \pi_s \), the expected number of steps needed to hit an optimal solution is exponential.

### 3.3 Metropolis on hidden line graphs

Let \( H_{n,p} \) be the class of hidden line graphs on vertex set \( V = \{1..n\} \) and where each possible edge \( \{i, j\} \) appears in the graph with probability 1 if \( |i - j| = 1 \) and probability \( p^{i-j} \) otherwise (\( p \leq 1/2 \)). Essentially, hidden line graphs are line graphs with a few additional edges. As line graphs and random linear graphs, the normal layout of hidden line graphs is very close to the optimum.

In the following, we consider the application of Metropolis to hidden line graphs. The reasoning follows the one in [12]. Metropolis begins at a random layout. A move is the random selection of a vertex in \( \varphi \) and its exchange of position with the left or right neighbor, so that a new arrangement \( \varphi' \) is derived (call it *neighbor flip*). A proposed move that changes the cost by a (positive or negative) quantity \( \delta = c_\varphi - c_{\varphi'} \) is accepted with probability \( A(\delta) = \frac{e^{-\delta/T}}{1 + e^{-\delta/T}} \). The acceptance rule defines a Markov chain [11].

Given a linear arrangement \( \varphi \) on a \( H_{n,p} \) graph, define its imbalance as \( \text{imb}(\varphi) = \sum_{i=1}^{n-1} |\varphi(i) - \varphi(i+1)| \). The imbalance is minimal and equal to \( n - 1 \) if and only if \( \varphi \) is exactly the normal layout of \( H_{n,p} \). The algorithm can compute \( c_\varphi \), but it cannot calculate \( \text{imb}(\varphi) \) because it does not know the labels of the nodes in the hidden line. We wish to analyze the progress of the imbalance.

Notice that our definition of neighbor flip moves just allows the imbalance to either stay the same, or change (increase or decrease) at most 4.

In the sequel we fix a \( T \geq \sqrt{n} \). Later we will need in fact \( T > n^2 \).

**Lemma 5** For any arrangement \( \varphi \) with imbalance \( k \), the probability that a proposed move decreases \( k \) by 2 is \( \Pr[k \text{ decreases}] = 1/2 - \mathcal{O}(k/n^2) \).

For a given layout \( \varphi \) with imbalance \( k \) and a move \( M \) that decreases \( k \), let \( X' \) be the random variable representing the cost decrease that occurs when move \( M \) is applied to \( \varphi \). Then \( A(X') \) is the probability that a given \( k \)-decreasing move is accepted.

**Lemma 6** Let \( T \geq \sqrt{n} \). Then, \( \mathbb{E}[A(X')] \geq \frac{1}{2} + \Omega \left( \min \left\{ \frac{kp}{4T}, 1 \right\} \right) \).
For a given arrangement \( \varphi \) with imbalance \( k \), let \( f^+(\varphi) \) (resp. \( f^-(\varphi) \)) be the random variable denoting the probability that after one move the new arrangement has imbalance \( k+2 \) (resp. \( k-2 \)).

By symmetry and the previous two lemmas, we have:

**Lemma 7** Let \( T \geq \sqrt{n} \). Let \( \text{imb}(\varphi) = k \). Then,

\[
E[f^+] \leq 1/4 - \Omega \left( \min \left\{ \frac{kp}{4T}, 1 \right\} \right) - O\left( \frac{k}{n^2} \right)
\]

\[
E[f^-] \geq 1/4 + \Omega \left( \min \left\{ \frac{kp}{4T}, 1 \right\} \right) - O\left( \frac{k}{n^2} \right).
\]

We say that \( \varphi \) is \( \lambda \)-atypical if either \( |f^+ - E[f^+]| > \lambda \) or \( |f^- - E[f^-]| > \lambda \).

**Lemma 8** \( \forall \varphi, \forall \lambda > 0 \), the probability over the class \( \mathcal{H}_{n,p} \) that \( \varphi \) is \( \lambda \)-atypical is at most \( 4 \exp\left(-\frac{32}{9} \lambda^2 T^2\right) \).

From here on we need \( T > n^{2+\epsilon} \). Starting from an initial imbalance \( k = \Theta(n^2) \), the bias \( f^- - f^+ = \Omega\left(kp/T\right) = \Omega(n^{-\epsilon}) \). This can be expected over \( n^{2+\epsilon} \) moves to contribute an imbalance \( \Omega(n^{-2}) \) and thus move very close to the hidden line. The exact analysis here is given by modeling the sequence \( k(t) \) as a random walk which has an absorbing point at \( k = n - 1 \) and a bias to move to the left with probability \( \Omega(n^{-\epsilon}) \). Thus we have:

**Theorem 3** On the average, starting from a random layout, the Metropolis algorithm will hit the normal layout in time \( O(n^{2+\epsilon} \log n) \).

### 4 Sequential and Parallel SS+SA Heuristics

In this section we first introduce the ideas that yield to the sequential SS+SA heuristic and then show two ways to parallelize it on a distributed memory machine with a few processors: the *exact* and *chaotic* methods. This heuristic is specially suited for geometric graphs. The basic idea of the SS+SA heuristic consists in building first a globally good layout by using SS [26] and after improving it locally running SA starting with a low temperature.

**A new neighborhood distribution: FlipN.** Since the SA process will start from a quite good solution at a low temperature, it can be expected that the number of rejected moves in the Flip2 neighborhood will be high. As a consequence, finding acceptable moves will be difficult. In order to reduce the time of this search, we can observe that on a good solution, changes that are worth to be tried must be close in the current layout. Figure 4 supports this affirmation: for each one of the \( n(n-1)/2 \) possible moves in the Flip2 neighborhood, we have computed their gain \( \delta \) and have accepted or rejected the move according to the SA criteria for different temperatures. The figure shows how many moves have been accepted in function of the distance \( |\varphi(u) - \varphi(v)| \) between nodes \( u \) and \( v \) in the layout \( \varphi \) obtained by SS. Clearly, the distributions follow a Gaussian tail and they show that moves involving close nodes will have more possibilities of being accepted.

This fact gives rise to the use of a new neighborhood relation, or more specifically, a new distribution for the Flip2 neighborhood: Just after obtaining the solution found by SS, we perform a sampling of the Flip2 neighborhood and compute the mean (\( \mu \)) and deviation
(σ) of the distances between acceptable moves (the size of this sampling has been fixed to be n√n). Then, during the SA process, moves will be generated producing pairs of vertices whose distance follows a normal distribution \( N(\mu, \sigma) \). We call this new neighborhood FlipN.

**Cooling schedule.** Selecting and tuning a cooling schedule has been always a key problem when using SA [13, 14]. The design of our cooling schedule is very pragmatisical, close to the classical ones and, influenced by previous experiments. We use a geometric cooling schedule that decrements the temperature at each round multiplying it by \( \alpha \) (close to 1). For each temperature, a round of \( r \) moves will be tried. If these moves reduce the objective function more than a ratio \( \rho \), we assume that this temperature has not yet reached equilibrium and so we run another round at the same temperature. The SA process ends when less than \( B \) moves in a round had a strictly positive gain.

**The sequential algorithm.** The concrete values of the free parameters we have used are \( \alpha = 0.95 \) for the cooling schedule, \( r = \beta n \mu \) moves per run with \( \beta = 10 \) as size magnifier, \( B = 10 \) as frozen detector, \( t_0 = 10 \) as starting temperature and \( \rho = 0.9999 \) for the equilibrium ratio.

```
function SS+SAG(G) is
    Generate an initial layout \( \varphi \) using Spectral Sequencing
    Sample the neighborhood at \( t_0 \) to obtain \( \mu \) and \( \sigma \)
    \( t := t_0; \ r := \beta \cdot \mu \cdot n \)
    repeat
        \( c := c_0; \ b := 0 \)
        repeat \( r \) times
            Select \( u, v \) with \( |\varphi(u) - \varphi(v)| \) drawn from \( N(\mu, \sigma) \)
            \( \varphi' := \varphi; \ \varphi'[u] := \varphi[v]; \ \varphi'[v] := \varphi[u]; \ \delta := c_\varphi - c_{\varphi'} \)
            with probability \( \min(1, e^{-\delta/t}) \)
            \( \varphi := \varphi' \)
            if \( \delta > 0 \) then \( b := b + 1 \)
        end with
        if \( c_{\delta, \varphi} / c > \rho \) then \( t := \alpha \cdot t \)
        until \( b < B \)
    return \( (\varphi, c_\varphi) \)
end
```

**Exact parallel SS+SA.** Before presenting the parallel algorithm, we need some definitions. Given an integer \( p \) representing the number of available processors, a layout \( \varphi \) on a graph \( G = (\{1..n\}, E) \) and an increasing sequence of indices \( j_0, j_1, \ldots, j_p \) such that \( j_0 = 0 \) and \( j_p = n \); define a \( p \)-partition \( V_1, V_2, \ldots, V_p \) of \( V \) by \( V_i = \{ u \in V \mid j_{i-1} < \varphi(u) \leq j_i \} \). Let \( V_0 \) be the \( p \)-partition induced by \( j_i = in/p \) (\( vi \in 1..n-1 \)), let \( V_{i+1} \) be the \( p \)-partition induced by \( j_i = in/2 - n/2p \) and let \( V_{-1} \) be the \( p \)-partition induced by \( j_i = in/2 - n/2p \). Given a \( p \)-partition, an edge is said to be a cut if it has its vertices in different partitions. Vertices that have an adjacent edge in the cut are said to be in the frontier, otherwise they are said to be free.

The parallel SS+SA begins by computing an initial solution \( \varphi \) using SS. This step is done sequentially, because it can be done very fast. Afterwards, we sample in parallel the Flip2 neighborhood of \( \varphi \) to obtain the values of \( \mu \) and \( \sigma \). At this moment the SA algorithm begins. The schedule is the same than in the sequential algorithm, but now the Metropolis
The basic idea of the data level parallelization of Metropolis consists in computing $p$-partitions of $V$ using the current layout $\varphi$ and concurrently apply moves within the FlipN neighborhood on these partitions, one partition per processor. The main part of the parallel SS+SA using $p$ processors is:

\begin{verbatim}
function Parallel SS+SA($G$) is
    (Alg. 2)
    Sequentially, generate an initial layout $\varphi$ using Spectral Sequencing
    In parallel, sample the neighborhood at $t_0$ to obtain $\mu$ and $\sigma$
    $t := t_0$; $r := \beta \cdot \mu \cdot n/4p$
    repeat
        $c := c_\varphi$; $b := 0$
        Metropolis(0); Metropolis(+1); Metropolis(0); Metropolis(-1)
        if $c_\varphi/c > \rho$ then $t := \alpha \cdot t$
    until $b < B$
    return $\langle \varphi, c_\varphi \rangle$
end
\end{verbatim}

The Metropolis function (see Alg. 3) takes full advantage of parallelism. At the beginning, each processors gets its own copy of the input layout. Each processor takes care of one of the $p$ partitions. In this partition, moves are generated on its own copy of the layout. If these moves are not forbidden (i.e., none of the vertices that it flips are in the frontier), they are accepted or rejected according to the Metropolis criterion. Notice that during this phase the information owned by each processor is maintained coherent with respect to the information stored in the other processors. A processor does not need to communicate its moves to the other processors, since they will never use this information. As a consequence, forbidding moves removes the need of an expensive communication process, while not affecting the correctness of the algorithm. After performing $r$ iterations, a synchronization phase between all the processors is performed. During this synchronization, the global layout is easily rebuilt through the combination of each processor own copy of the layout and the $p$-partition. The high level algorithm is as follows:

\begin{verbatim}
function Metropolis($x$) is
    (Alg. 3)
    for each processor $i \in 1..p$ do in parallel
        Get a private copy of layout $\varphi$
        From $V_x$ compute $V_i$ and compute the frontier nodes
        (* Perform Metropolis in $V_x$ *)
        $b_i := 0$
        repeat $r$ times
            Select $u, v$ in $V_i$ with $|\varphi(u) - \varphi(v)|$ drawn from $\mathcal{N}(\mu, \sigma)$
            if $u$ and $v$ are free wrt $V_i$ then
                $\varphi' := \varphi$; $\varphi'[u] := \varphi[v]$; $\varphi'[v] := \varphi[u]$; $\delta := c_\varphi - c_{\varphi'}$
                with probability $\min(1, e^{-\delta/\beta})$ do
                    $\varphi := \varphi'$; if $\delta > 0$ then $b_i := b_i + 1$
                end with
            end if
        end repeat
        (* Synchronize and Gather data *)
        Rebuild a global layout $\varphi$ from the ones distributed among procs according to $V_x$
        $b := b + \sum_{i=1}^{p} b_i$
    end for
end
\end{verbatim}
The basic reason for this parallel heuristic to work is that on sparse geometric graphs, partitions induced by a good layout have a few cutting edges. As a consequence, only a few moves inside a partition will be forbidden. So, in each call to the Metropolis process there will be many opportunities to optimize the individual partitions. The trick of using three different partitions intercalated in four phases is used in order to avoid forbidding always the same moves. Otherwise we would obtain layouts which would be well arranged inside each partition, but locally bad near the frontiers. With respect to the efficiency of the algorithm on real machines, for an enough large \( r \), the time spent in the synchronization, broadcasting and gathering phases is low with respect to the computing phases. Thus, large speedups and high processor use can be expected, as far as many nodes remain free.

Chaotic parallel SS+SA. Whereas forbidding moves enables the processors to have always an up-to-date information, it restricts the possibilities of optimizing. If we remove this restriction (i.e. we allow processors to freely move frontierer nodes), as moves are applied into a concrete partition, the global state of the system still would be coherent, since it represents a valid layout. However, after the move, other processors would compute \( \delta \) with an out-of-date information, committing an error. Intuitively, this error should not be very high, and would decrease as the temperature is lowered. A way to reduce the error is to perform frequent synchronizations in which all the processors perform an all-to-all communication in order to gossip the more recent up-to-date layout. A trade-off between this additional communication time and error precision must be done.

5 Experimental Evaluation

In this section we present, compare and analyze some empiric results aiming to evaluate and tune the performance of our heuristics. Appendix B describes the experimental conditions.

The benchmark graphs. We always worked with large graphs (more than 4000 nodes). The main characteristics of our benchmark graphs are shown in Table 1. Since our aim is to measure the heuristics in sparse and geometric graphs, we have selected some graphs belonging to the following families:

- Geometric random graphs \( G_{r_-,r_+} \): 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 between \( r_- \) and \( r_+ \). The geometric graph has been generated using as parameters \( n = 5000 \), \( r_- = 0.027 \) and \( r_+ = 0.04 \).
- Thin geometric graphs \( G_{\sqrt{n},p,r} \): The vertices are points of a \( \sqrt{n} \times \sqrt{n} \) square grid. Each point is selected as a vertex with probability \( p \). There is an edge between all vertices whose (Euclidean) distance is at most \( r \). For thingeo, \( \sqrt{n} = 125 \), \( p = 0.33 \) and \( r = 4 \).
- Random linear graphs: \( rlg \) belongs to \( L_n \) with \( n = 5000 \).
- Graphs from finite element discretizations: airfoil1, whitaker3 and 4elt.

Most of the experimental results are only shown for the airfoil1 graph. Unless otherwise stated, the same behaviors were observed on the rest of the benchmark graphs.
Analysis of the SS heuristic. In order to verify the good behavior of the Spectral Sequencing heuristic, we applied it to our benchmarking graphs. Figure 5 compares SS with some other heuristics that were described in [26]. For all the graphs, the results obtained show that SS is clearly the heuristic that provides better approximations. Moreover, it is considerably faster than the other heuristics.

The effect of using SS to start SA. Figure 6 compares two annealing curves of SA on the airfoil11 graph, one of them starting with the SS solution and the other starting with a randomly generated layout (both use the Flip2 neighborhood). The same behavior was observed on the rest of our benchmark graphs. It is clear that the use of SA with an starting solution produced by SS improves the quality of the final solution and decreases dramatically the runtime. Moreover, by using a random solution, it seems difficult (or at least a very large amount of time would be necessary) to achieve a quality similar to the one obtained by SS.

Comparing FlipN with Flip2. The next two experiments were designed to evaluate the effect of using the FlipN neighborhood instead of the more traditional Flip2, i.e. to favor moves among close nodes in the current layout with respect to try to flip the labels of any pair of nodes. For the particular case of the airfoil11 graph, Figure 7 shows a trace of the approximation in function of the time when using the two different neighborhood distributions. The measuring of the proportion of accepted movements at each temperature is shown in Figure 8. It is clear that the new neighborhood distribution increases the ratio of accepted movements; while at temperatures greater than 1 Flip2 is only able to propose less than 0.1% of accepted moves, the number of accepted moves produced by FlipN range from 2 to 10%. However, when using FlipN, a big part of the moves have a null gain ($\delta = 0$). Even if these moves do not improve the solution, they can increase the possibilities of SA to quickly escape from plateaus or local minima.

Quality and time of the parallel SS+SA heuristics. We have measured the quality of the solution obtained by our parallel heuristics as well as the time required to compute them. In order to enable a comparison with respect to the sequential heuristic, we define the time efficiency (also called processor utilization) and solution ratio terms. Denote by $T$ the time needed to execute the sequential SS+SA heuristic and by $C$ the cost obtained, let $T_p$ be the time of the parallel SS+SA heuristic ran on $p$ processors and $C_p$ the cost obtained; then the time efficiency is $T/(pT_p)$ and the solution ratio is $C/C_p$. For each graph, Table 2 shows the time efficiency and solution ratio obtained. From these results, it can be observed that:

- Time efficiency is always close to the 100% (except for r1g). This fact shows that we achieve an efficient use of the parallel system, because the speedup is close to the number of processors used. (Notice that time efficiencies greater than 100% are due to the random nature of the algorithms.)
- Solution ratio is always close to the 100% (except for r1g). This fact shows that the parallel heuristics deliver solutions of comparable quality to the ones delivered by the sequential program.
- In general, chaotic parallel SS+SA gives slightly better solution than exact parallel SS+SA. In the case of the r1g graph, this difference very big.
The solution ratio of exact parallel SS+SA decreases as the number of processors increases. On the contrary, the chaotic parallel SS+SA offers solutions of similar quality independently of the number of processors. Thus, chaotic parallelization offers better scalability than exact parallelization in terms of solution quality. It also worth to remark the poor performance of exact SS+SA on rlg. The reason is that not many of its nodes are on the frontier.

We also observed (see Figures 9, 10, 11) that the random graphs (rlg, geometric and thingeo) have some more noisy annealing curves than the "real-life" graphs arising from finite elements (airfoil1, whitaker3 and 4elt). Differences between the approximation of different kinds of graphs was also previously remarked in [26] for the MinLA problem.

6 Conclusions

We have presented three types of quasi-linear graphs and have studied some of their properties. These classes of graphs share the fact that we know their optimal layouts and can formulate reasonings on the behavior of Local Search algorithms applied to them. On the positive side, we have shown that Metropolis can efficiently discover the canonical layout of hidden line graphs. On the other hand, we have presented an instance for which Hill Climbing can be fooled into an exponential number of steps.

We have also presented new sequential and parallel heuristics for the MinLA problem on geometric graphs that combines Spectral Sequencing (SS) with Simulated Annealing (SA). The aim was to accelerate a previous SA algorithm to approximate this problem. Doing so, we have addressed three main points in the improvement of simulated annealing techniques: the use of better-than-random initial solutions, the use of an adequate neighborhood specially adapted to improve good solutions and its parallelization on a distributed memory machine by partitioning data among processors.

The extensive experimentation and the benchmarkings we have presented on large sparse graphs shows the viability and practicality of our approaches based on SS+SA. We found better approximations than in previous works [26] in radically less time, and the parallel heuristics have an excellent behavior in terms of running time and solution quality compared to the sequential heuristic.

Availability of code, instances and raw data. In order to help to reproduce and verify the measurements and the programs mentioned in this research, its code, instances and raw data are available at http://wwwlsi.upc.es/~jpetit/MinLA/SS+SA on the World Wide Web.

Acknowledgements. The authors thank María José Serna for enlighting discussions and Spyros Kontogiannis for his help. The implementation and evaluation of these heuristics has been done at the C4 (Centre de Computació i Comunicacions de Catalunya).
References


Appendix: Proofs of Section 3

Proof of Lemma 1. The expected cost of the normal layout is given by $\mu = \mathbb{E}[C_{G,\psi}] = \frac{n^2}{2} + O(n \log n)$. Label the $m = n(n-1)/2$ possible edges $\{i, j\}$ ($1 \leq i < j \leq n$) following some order $e_1, e_2, \ldots, e_m$. Define $I_{ek} = 1$ when $e_k \in E$ and $I_{ek} = 0$ when $e_k \notin E$. Consider the Doob Martingale defined by the sequence $X_0, X_{e_1}, X_{e_2}, \ldots, X_{e_m}$ with $X_{e_k} = \mathbb{E}[C_{G,\psi} \mid \mathcal{F}_{e_k}]$, where $\mathcal{F}_{e_k}$ is the filter defined by fixing the first $k$ bits in $I_{e_1}, \ldots, I_{e_m}$. Notice that for any $e_k = \{i, j\}$, $|X_{e_k} - X_{e_{k-1}}| = \frac{j-i}{j-i+1} < 1$, therefore we can apply Azuma’s inequality [23] to obtain that

$$\forall \lambda > 0 : \Pr [|C_{G,\psi} - \mu| \geq \lambda] = \Pr [|X_{e_m} - X_0| \geq \lambda] \leq 2 \exp \frac{-\lambda^2}{n^2 - n}. \quad \square$$

Proof of Lemma 3. The cost of a layout on $I_n$ is the difference of numberings between adjacent vertices. A different way to look at the problem is to keep fixed in consecutive order the layout numbers and change the edges. The graph associated with the layout $\pi_1 = (7, 4, 8, 2, 6, 1, 3, 5)$ is shown in Figure 1a (the first node of $I_7$ gets value 7, the second gets 4 and the last one gets 5), notice $c(\pi_1) = 26$. In Figure 1b, we can see the two minimal layouts on $\pi_1^*$ and $\pi_2^*$ with $c(\pi_1^*) = c(\pi_2^*) = 7$. In Figure 1c we have the graph for $\pi_2 = (1, 2, 7, 6, 3, 4, 5)$ with cost $c(\pi_2) = 12$. We denote the graphs with crossings like the one in Figure 1a “turtle” graphs, and the graphs with nested layouts “snail” graphs. Except the two optimal layouts, every layout on $I_n$ can be represented as a snail, a turtle or a combination of both. Notice the snail could be a large line with a small shell, or a large nested collection of shells. In a snail graph we have the shells which are the curves between vertices with assignment differing by more than 1, the lines which are consecutive ordered line of vertices, the two lines ending in a vertex of degree 1 are the terminal lines. The elbows are the vertices joining lines and shells. For instance, in Figure 2, the edges $(2, 32), (3, 27), (8, 26), (9, 21), (14, 20)$ are shells; the terminal lines are $(1, 2)$ and $(15, 16)(16, 17) \ldots (19, 20)$, the elbows are $2, 3, 8, 9, 14, \ldots$

Given two layouts $\pi$ and $\pi'$ we say that they are neighbors if $\pi^{-1} \pi'$ is a transposition. We will denote it $|\pi \circ \pi'| = 1$. (In other words, if $|\pi \circ \pi'| = 1$ we can go from one to the other by flipping two elements). Given a layout $\pi$, let $N_\pi = \{\pi' : |\pi \circ \pi'| = 1\}$ be its set of neighbors. Notice $|N_\pi| = \frac{n^2 - n - 2}{2} = O(n^2)$. Consider the usual permutation graph $S$ with vertex set $\mathcal{S}_n$ and any two $\pi$ and $\pi'$ are connected by an edge iff $|\pi \circ \pi'| = 1$ [20]. For labels $i, j$ let $i \leftrightarrow j$ denote the exchange of edges between nodes $i$ and $j$ (equivalent to flipping labels $i$ and $j$). Any layout of $I_n$ corresponds to a permutation in the symmetric group $\mathcal{S}_n$. From now on we shall denote by $\pi \in \mathcal{S}_n$ any layout, and $\pi^*$ the identity permutation corresponding to $\varphi^*$.

Let us define the following transition process on $S$, given by flipping two numbers in the layout. Let $M_{\pi_i} = \{\pi_j \in N_{\pi_i} \mid c(\pi_j) \geq c(\pi_j)\}$. From state $\pi_i$:

1. With probability $1/2$ stay in $\pi_i$.
2. Select $\pi_j$ according to $P(\pi_j) = \begin{cases} \frac{2}{n^2 - n} & \text{if } \pi_j \in N_{\pi_i} \\ 0 & \text{otherwise} \end{cases}$
3. If $\pi_j \in M_{\pi_i}$ go to $\pi_j$, else stay in $\pi_i$.

The previous process is a HC algorithm for MINLA on $I_n$ with the Flip2 neighborhood [11].

The proof of Lemma 3 will be sketched in several steps. First we shall sketch the proof that for any $\pi_i$ different than $\{\pi_1^*, \pi_2^*\}$, we have $|M_{\pi_i}| \geq 1$. 13
For any non-minimal layout $\pi_i$ its graph either contains turtles, snails or both. If it contains a turtle, the flipping of the values of any crossing produces a $\pi_j \in M_{\pi_i}$, (for instance in Figure 1a, the flipping $4 \leftrightarrow 6$ produces a $\pi_j$ with cost 17, while the flipping $2 \leftrightarrow 3$ produces a $\pi_j$ of cost 24). In case the graph contains snails, flip the elbow of the inner line with its neighbor. The cost of the new graph is the same.

The previous lemma does not say anything about the possibility of getting stuck in a plateau. We shall prove that from any non-minimal layout $\pi_i$, there exists a finite set of transitions that will go to a layout $\pi_j$ with $c(\pi_i) > c(\pi_j)$.

If the layout graph contains turtles, it is not difficult to see (by cases) that we can break all crossings until either arriving to a snail or to lower the cost (if the crossing involves at least one node of cardinality 1 or one node with both edges in the direction of the crossing). When having a snail, we can eliminate one by one all the shells, starting by the inner one. If we have a shell with a terminal line of length $l$, we go sliding the elbow until decreasing the cost by 1 (it will take $l - 1$ steps). Repeat the operation until converting the shell in a inner line of the next shell, and it would take $\frac{l^2 + l}{2}$ steps and it will decrease the total cost by $l$.

For instance, in Figure 1c, flip 3 $\leftrightarrow$ 4 and then 4 $\leftrightarrow$ 5 to decrease the cost by 1, (the new shell will be between 3 and 5) then 3 $\leftrightarrow$ 4 again to convert the inner line and shell 3–6 into the line 3–4–5–6–7.

Thus we have a proof of lemma 3.

**Proof of Lemma 4.** Consider a layout $\pi_s$ of $\mathcal{L}_n$ consisting of a turtle with $n/5$ nested shells, each with a line of length 5 (see Figure 2 for an example with $n = 32$). Any sequence of graphs to arrive to the optimal passes by "flattering" all the shells. The optimal way to do it is starting with the inner shell and convert it into a line in the manner described previously. The number of flips needed for the whole process is

$$\sum_{j=0}^{n/5} \sum_{i=1}^{(j+1)5+j} i = \frac{6}{125} n^3 + \mathcal{O}(n^2).$$

For instance the optimal sequence in Figure 2 is: 20 $\leftrightarrow$ 19, 19 $\leftrightarrow$ 18, 18 $\leftrightarrow$ 17, 17 $\leftrightarrow$ 16, 16 $\leftrightarrow$ 15, now the cost will be one less and the new inner shell will be between 15 and 20. Again we repeat the sliding of the new elbow 15 until getting a new inner shell (16, 20). After 15 flippings we have converted the inner shell in a line 9, 10, 11, ..., 19, 20, with new inner shell (9, 21). At the end, after 1295 flippings, we will hit the optimal layout.

Notice that the probability that starting from $\pi_s$ we arrive to an optimal MinLA in the exact number of steps is less than $\mathcal{O}(\frac{1}{n^1})$. Moreover, if we start from the layout $\pi_s$, the size of its plateau is $\mathcal{O}(\frac{5^n}{n^3})$. Recall that a plateau is the set of accessible layouts with exactly the same cost (see e.g. [15]). Every layout in the plateau has degree $\mathcal{O}(n/5)$. Among all the layouts in the plateau, $\mathcal{O}(5^{\frac{n^2}{5}})$ of them go to a layout with less cost, let us call them gate layouts. Therefore to exit from the plateau the HC should arrive to one of these layouts. Using standard theory of random walks [19], the expected hitting time to arrive to one of the gate layouts, starting from $\pi_s$, and moving randomly using the HC heuristic is $\Theta(5^{\frac{n^2}{5}})$. If the expected time to get out of the first plateau is exponential, we have proved Theorem 2.

**Proof of Lemma 5.** Consider that we choose to move node labeled $x$ in the $\mathcal{H}_{n,p}$. In figure 3, look at positions of the nodes labeled $x - 1$ and $x + 1$. Clearly $k$ stays the same when
the initial and final positions of \(x\) are between \(x - 1\) and \(x + 1\). The imbalance decreases when \(x\) moves closer to the interval and increases when \(x\) moves away. The rest is easy. There are \(n\) possibilities for \(x\), \(n\) possibilities for given positions of \(x - 1\), \(x + 1\) and we must sum over all final positions so that \(x\) goes away from the interval. Because of symmetry, \(k\) is decreased in at least half the cases, except inside the interval.

\[\square\]

\textbf{Proof of Lemma 6.} The distribution of \(X^t\) can be expressed as the sum and difference of \(n - 1\) Bernoulli trials, corresponding to \(n - 1\) potential edges affected by the specified move. Create a symmetric around the origin \(X^0\), again as sum/difference of \((n - 1)\) Bernoulli trials. Clearly, \(X^0 - X^t = \mathcal{O}(k \cdot B_p)\) (in fact \(X^0 - X^t = \mathcal{O}(kB_p + kB_p^2 + \ldots) = \mathcal{O}(kB_p)\)). Thus, \(\Pr[X^0 - X^t \geq kp/2] \geq 1/2\) while \(\Pr[|X^0| \leq 2\sqrt{n}] \geq 3/4\). Thus, with probability at least 1/4 the events \("X^0 - X^t \geq kp/2"\) and \("|X^0| \leq 2\sqrt{n}\) occur together (independently of any probabilistic correlation among them). So, since \(T \geq \sqrt{n}\), with probability at least 1/4, the events \(|\frac{X^0}{T}| \leq 2\) and \(\frac{X^0}{T} - \frac{X^t}{T} \geq \frac{kp}{4T}\) occur together. In this event, \(A(X^t) - A(X^0) = \Omega(\min\{\frac{kp}{4T}, 1\})\) since in any bounded range the sigmoid \((1 + e^x)^{-1}\) has negative gradient uniformly away from zero. Then,

\[E[A(X^t) - 1/2] = E[A(X^t) - A(X^0)] \geq \Omega\left(\min\left\{\frac{kp}{4T}, 1\right\}\right)\]

which proves the lemma.

\[\square\]

\textbf{Proof of Lemma 8.} \(f^-\) is the sum over the probabilities of \(k\)-decreasing moves (flips of neighbors) that the move is both proposed and accepted. Take a node \(x\). The probability of proposing a flip of \(x\) is \(1/2n\) (1/n at end). The probability to accept it depends on the change in sum of cuts. For any particular edge out of \(x\) the cost change is at most 1 and each such \(x\) can have at most 3 edges out of it expectedly (2 for the base line and \(p + p^2 + p^3 + \ldots = \mathcal{O}(p) < 1\) for all the rest). Since the derivative of \(A(x)\) is at most \(1/4T\) we get for the whole change (expectedly) at most \(1/2n \cdot 3 \cdot 1/4T = \frac{3}{8nT}\). Summing over all possible edges we get

\[
\left(\frac{n}{2}\right) \left(\frac{3}{8nT}\right)^2 \leq \frac{9}{16T^2}
\]

and the generalized Azuma’s Inequality produces the result.

\[\square\]

\textbf{B Appendix: Experimental Conditions}

The sequential SS+SA heuristic has been coded in the C++ programming language. The parallel heuristics use C++ augmented by the MPI message passing library. All the programs have been run on an IBM SP2 computer with 8 Power2 processors under the AIX operating system. This computer has enough main memory to run our programs without delays due to paging. Moreover, its interconnection network provides high performance communication between the processors and the MPI implementation for the SP2 fully profits it.

All the experiments have been executed with processors and network in exclusive mode (expect for system daemons) and measure the total elapsed (wall-clock) time. Pre- and post-processing times are included in our measures (except the time used to read the input graph). The programs have been compiled using maximum optimization compiler options and have
been tested using two different random number generators without noticing any anomaly due to them.

The computation of the $x^{(2)}$ vector has been implemented using the Chaco library [10].

C Appendix: Figures used in the Proofs

![Figure 1: Some example of layouts on $I_n$ graphs.](image)

![Figure 2: A difficult instance for Hill Climbing](image)

![Figure 3: The changes of the imbalance due to a new move.](image)
D Appendix: Summary of the Experimental Results

Figure 4: Number of accepted moves in function of their distance and the temperature (t) on the solution found by Spectral Sequencing for the airfoil1 graph. (t < 0 means that only strictly descendent moves are accepted.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Nodes</th>
<th>Edges</th>
<th>Degree</th>
<th>LB</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometric</td>
<td>5000</td>
<td>32983</td>
<td>2/13.2/28</td>
<td>135092</td>
<td>2205552</td>
</tr>
<tr>
<td>thingeo</td>
<td>5167</td>
<td>39924</td>
<td>3/15.4/27</td>
<td>182589</td>
<td>1701551</td>
</tr>
<tr>
<td>rlg</td>
<td>5000</td>
<td>35414</td>
<td>3/14.1/26</td>
<td>6773813</td>
<td>12171878</td>
</tr>
<tr>
<td>airfoil1</td>
<td>4253</td>
<td>12289</td>
<td>3/5.78/10</td>
<td>24220</td>
<td>278253</td>
</tr>
<tr>
<td>whitaker3</td>
<td>9800</td>
<td>28989</td>
<td>3/5.91/8</td>
<td>57824</td>
<td>1169065</td>
</tr>
<tr>
<td>4elt</td>
<td>15606</td>
<td>45878</td>
<td>3/5.9/10</td>
<td>142051</td>
<td>2241776</td>
</tr>
</tbody>
</table>

Table 1: Benchmark graphs. For each graph, its name, number of nodes (n), number of edges (m), degree information (minimum/average/maximum). The column labeled LB shows the best known lower bound and Best the cost of the best known arrangement. The cost of the normal arrangement of rlg is 12308949.
Figure 5: Comparison of previous heuristics on the \texttt{airfoil1} graph: random (\texttt{random}), variations of greedy successive augmentation algorithms (\texttt{gre, rbs, bfs dfs}), variations of HC (hillcE, hillc2, hillc3) and SS (ss). The boxplots represent the distribution of the approximations found in 1000 independent experiments and the average time needed.

Figure 6: Annealing curve on the \texttt{airfoil1} graph when using a random solution or the SS solution as initial solutions and the Flip2 neighborhood. Each unit of time corresponds to 2 minutes.
<table>
<thead>
<tr>
<th>Processors</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometric</td>
<td>100.000</td>
<td>97.008</td>
<td>100.052</td>
<td>108.042</td>
<td>106.657</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>100.824</td>
<td>99.587</td>
<td>97.359</td>
<td>92.755</td>
</tr>
<tr>
<td>thingeo</td>
<td>100.000</td>
<td>101.241</td>
<td>107.729</td>
<td>110.952</td>
<td>106.243</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>99.926</td>
<td>99.472</td>
<td>97.905</td>
<td>95.336</td>
</tr>
<tr>
<td>rlg</td>
<td>100.000</td>
<td>121.597</td>
<td>156.353</td>
<td>143.626</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>88.873</td>
<td>80.993</td>
<td>80.669</td>
<td></td>
</tr>
<tr>
<td>airfoil1</td>
<td>100.000</td>
<td>99.487</td>
<td>90.888</td>
<td>98.343</td>
<td>97.995</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>99.691</td>
<td>99.921</td>
<td>98.899</td>
<td>97.441</td>
</tr>
<tr>
<td>whitaker3</td>
<td>100.000</td>
<td>107.733</td>
<td>100.893</td>
<td>109.053</td>
<td>106.674</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>99.901</td>
<td>100.017</td>
<td>99.875</td>
<td>99.475</td>
</tr>
<tr>
<td>4alt</td>
<td>100.000</td>
<td>99.503</td>
<td>98.691</td>
<td>99.154</td>
<td>99.257</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>99.640</td>
<td>99.478</td>
<td>99.313</td>
<td>98.818</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processors</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometric</td>
<td>100.000</td>
<td>96.260</td>
<td>100.755</td>
<td>97.782</td>
<td>105.714</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>100.834</td>
<td>100.405</td>
<td>101.170</td>
<td>99.615</td>
</tr>
<tr>
<td>thingeo</td>
<td>100.000</td>
<td>111.238</td>
<td>101.377</td>
<td>105.641</td>
<td>99.598</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>100.021</td>
<td>99.660</td>
<td>100.039</td>
<td>100.199</td>
</tr>
<tr>
<td>rlg</td>
<td>100.000</td>
<td>95.176</td>
<td>98.807</td>
<td>102.679</td>
<td>97.298</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>99.998</td>
<td>100.002</td>
<td>100.001</td>
<td>100.000</td>
</tr>
<tr>
<td>airfoil1</td>
<td>100.000</td>
<td>97.731</td>
<td>94.908</td>
<td>106.972</td>
<td>92.448</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>99.560</td>
<td>100.680</td>
<td>99.625</td>
<td>99.913</td>
</tr>
<tr>
<td>whitaker3</td>
<td>100.000</td>
<td>96.385</td>
<td>107.085</td>
<td>106.222</td>
<td>109.766</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>100.024</td>
<td>99.895</td>
<td>99.849</td>
<td>100.028</td>
</tr>
<tr>
<td>4alt</td>
<td>100.000</td>
<td>101.900</td>
<td>104.372</td>
<td>100.329</td>
<td>96.198</td>
</tr>
<tr>
<td></td>
<td>100.000</td>
<td>99.753</td>
<td>99.447</td>
<td>99.572</td>
<td>99.855</td>
</tr>
</tbody>
</table>

Table 2: Time efficiency and solution ratio for exact and chaotic parallel SS+SA relatives to sequential SS+SA. For each graph, the top number is the time efficiency (%) and the bottom number is the solution quality (%).
Figure 7: SS+SA on airfoil1 using the Flip2 or FlipN neighborhoods: Evolution of cost in function of time (in seconds).

Figure 8: SS+SA on airfoil1 using the Flip2 or FlipN neighborhoods: Percent of accepted moves in function of temperature (the time goes right to left).
Figure 9: Exact and Chaotic Parallel SS+SA on geometric and thiggeo.
Figure 10: Exact and Chaotic Parallel SS+SA on rig and airfoil1.
Figure 11: Exact and Chaotic Parallel SS+SA on whitaker3 and 4eit.


LSI-98-4-R “BayesProfile: application of Bayesian Networks to website user tracking”, Ramón Sangüesa and Ulises Cortés.


LSI-98-7-R “Trust Values for Agent Selection in Multiagent Systems”, Karmelo Urzelai.

LSI-98-8-R “The use of SAREL to control the correspondence between Specification Documents”, Núria Castell and Àngels Hernández.

LSI-98-9-R “Intervalizing colored graphs is NP-complete for caterpillars with hair length 2”, C. Álvarez, J. Diaz and M. Serna.


LSI-98-12-R “Height-relaxed AVL rebalancing: A unified, fine-grained approach to concurrent dictionaries”, Luc Bougé, Joaquim Gabarró, Xavier Meseguer and Nicolas Schabanel.

LSI-98-13-R “HyperChromatic trees: a fine-grained approach to distributed algorithms on RedBlack trees”, Xavier Meseguer and Borja Valles.


Hardcopies of reports can be ordered from:

Núria Sánchez
Departament de Lenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord, Mòdul C6
Jordi Girona Salgado, 1-3
08034 Barcelona, Spain
secreslsi@lsi.upc.es

See also the Department WWW pages, http://www-lsi.upc.es/