Heuristics for the MinLA Problem: An Empirical and Theoretical Analysis

Josep Díaz
María José Serna
Jordi Petit i Silvestre
Paul Spirakis

Report LSI–98–42–R

Heuristics for the MinLA Problem: An Empirical and Theoretical Analysis*

(Extended Abstract)

Josep Díaz Jordi Petit i Silvestre María José Serna

Departament de Llenguatges i Sistemes Informàtics (LSI)
Universitat Politècnica de Catalunya (UPC)
Campus Nord — Mòdul C6
Jordi Girona Salgado, 1-3
08034 Barcelona, Catalonia (Spain)
Fax: +34 934 017 014
{diaz,jpetit,mjserna}@lsi.upc.es

Paul Spirakis

Computer Technology Institute (CTI)
Kolokotroni 3
GR 262 21
Patras (Greece)
spirakis@cti.gr

*Supported by ESPRIT LTR Project no. 20244 — ALCOM-IT, CICYT Project TIC97-1475-CE, DGES Project PB9J-0787-Koala and CIRIT 1997SGR-00366. The implementation and evaluation of these heuristics has been done at the $\mathcal{C}^4$ (Centre de Computació i Comunicacions de Catalunya).

1 Introduction

Several well-known optimization problems on graphs can be formulated as Linear Arrangement Problems. Their goal is to find a layout (ordering) of the nodes of an input graph such that a certain function is minimized. Linear arrangement problems are an important class of problems with many different applications in Computer Science, Biology, Archaeology, Linear Algebra, etc. Finding an optimal layout is **NP**-hard in general, and therefore it is natural to develop and analyze heuristics that give good approximations in practice.

In this paper we are concerned with the *minimum linear arrangement* problem (MinLA). Given an undirected graph $G = (V, E)$ with $n = |V|$ nodes, a *layout* is a bijection φ between V and the set $\{1, \ldots, n\}$. The goal of the MinLA problem is to find a layout that minimizes

$$C(\varphi) = \sum_{uv \in E} |\varphi(u) - \varphi(v)|.$$

Graphs encoding circuits, grids, finite element discretizations (FEM) or computer networks are typical instances of linear arrangement problems. We consider these instances as sparse graphs that have clustering and geometric properties (see Figure 7). For these interesting classes 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 several sequential heuristics to approximate MinLA and has measured their behavior on a set of benchmarking graphs [22]. The main conclusion was that for the type of graphs we are interested in, the best results were obtained with *Spectral Sequencing* (SS) and *Simulated Annealing* (SA). It was found, however, that the time required by SA to attain the same quality than SS was inordinate.

Contribution. In this paper, we address the issue of improving these heuristics. To do so, we first present a new heuristic called SS+SA based on the combination of SS and SA. The SS+SA heuristic can be efficiently parallelized on distributed memory machines (DMM). The basic idea of this new heuristic consists in building first a globally good layout by using SS and afterwards improving it locally through SA. The SA process is started at a low temperature and uses a special neighborhood. We analyze two methods to parallelize this phase.

In order to analyze SS+SA empirically, we present computational experiments on a set of benchmarking graphs made of large sparse clustered graphs. The results obtained show that the new heuristic clearly improves the quality and efficiency of previous proposed heuristics. Moreover, one of its parallel versions has an excellent behavior both in terms of running time and solution quality compared to the sequential counterpart. Therefore, for our benchmarking set, we show that the global strength of Spectral Methods combined with Local Search leads to an heuristic significantly better than either alone, and that parallelism can speedup this intensive computation on real machines without loosing too much efficacy.

On the theoretical side, we present two results. First, we present two new classes of random graphs that are useful to generate MinLA instances for which a good estimate of the optimum is known by construction. These classes of graphs are of particular interest to test and benchmark heuristics. Then, and as a negative result, we prove that *Hill Climbing* (HC) is not able to find in polynomial time the optimal layouts of a simple line graph even though it is guaranteed it will always hit these optima. Recall that Hill Climbing corresponds to SA at a zero temperature.

Relation to previous work. Even though the MinLA problem is **NP**-complete [8], there exist exact solutions for hypercubes, meshes and trees in polynomial time [9, 18, 1, 25, 3]. The lack of efficient exact algorithms for general graphs has given rise to the possibility of finding approximation algorithms. A polynomial approximation scheme for MinLA on dense graphs is presented in [7] and a $\mathcal{O}(\log n \log\log n)$ approximation using spreading metrics for general graphs is given in [6]. This last result was recently improved to a $\mathcal{O}(\log n)$ approximation factor for general graphs and a $\mathcal{O}(\log\log n)$ factor for planar graphs [23]. These algorithms [6, 23] have the disadvantage of having to solve a linear program with an exponential number of constraints using the Ellipsoid method, and thus are unsuitable for large graphs.

The approximation properties of sparse random graphs for different linear arrangement problems (including MinLA) are considered in [4], where it is proven that sparse instances drawn from the standard $\mathcal{G}_{n,p}$ distribution can be, with high probability, easily approximated. In this paper, we consider a different class of instances, so these results do not apply.

The SS heuristic for MinLA was originally proposed in [14]. Techniques to find lower bounds are reported in [14, 16]. The performance of these lower bounding techniques together with the analysis of many different heuristics (Greedy, Local Search —including HC, Metropolis and SA— and SS) on a set of benchmarking graphs is empirically studied in [22]. The design of the SS+SA heuristic was heavily based on this previous work.

A few works have succeed in proving the benefits of SA for particular classes of random instances (see e.g. [11, 20]). The limitations of SA have also been studied (see e.g. [24]). Our proof on the slow convergence of HC for MinLA on a simple input gives a new insight on these limitations: The fitness landscape of the search space can have large plateaus from which HC can hardly escape.

Finally, let us recall that the only know families of graphs for which their optimal solution is known is quite small and restricted to non random graphs. In this paper we enlarge this family of graphs with two different random classes whose minima are far from the average.

Organization. The rest of this paper is organized as follows. Section 2 presents the sequential SS+SA heuristic, after reviewing the SS and SA heuristics for MinLA. In section 3, we show two methods to parallelize SS+SA. Section 4 presents experimental results that evaluate the proposed heuristics on sequential and parallel environments. Theoretical results related with benchmarking are presented in Section 5, where we introduce two classes of random graphs. Finally, in Section 6, we present some theoretical considerations on HC. An Appendix with figures and tables completes the paper.

2 The Sequential SS+SA Heuristic

In this section we first introduce the ideas that yield to the sequential SS+SA, which consists in building first a globally good layout by using SS and afterwards improving it locally through SA started at a low temperature.

Spectral Sequencing. *Spectral Sequencing* (SS) is a particularly appropriated heuristic for some cases of MinLA [14]. Let $G = (\{1, \ldots, n\}, E)$ 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 λ_2 be the second smallest eigenvalue of L_G and $x^{(2)}$ its eigenvector (known as *Fiedler vector*). SS computes $x^{(2)}$ and after ranks each position of $x^{(2)}$. Thus, the heuristic returns an arrangement φ satisfying $\varphi(u) \leq \varphi(v)$ whenever $x_u^{(2)} \leq x_v^{(2)}$.

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. [26]). The computational intensive part of SS is the computation of the Fiedler vector, which is usually implemented using the Lanczos algorithm [21, 10].

The SS+SA heuristic uses SS in order to obtain a globally good first initial solution.

Local Search. Local Search algorithms improve iteratively an initial solution by performing local changes on its combinatorial structure. Usually, the initial solution is generated at random, but SS+SA will use the solution generated by SS. The basic idea of *Hill Climbing* (HC) is to improve iteratively the initial solution by performing local changes on its combinatorial structure. Only the changes that improve or keep the objective function are accepted. One way to avoid being stuck in poor quality local optima is to accept with small probability downhill moves. Metropolis [17] proposed an heuristic parameterized by a temperature t where a move that produces a gain of δ is accepted with probability $\min\{1, e^{-\delta/t}\}$. A refined heuristic is *Simulated Annealing* (SA), which consists of a sequence of runs of Metropolis with a progressive decrement of the t parameter [15]. A key point to apply SA to a given problem is the selection of its parameters such as the neighborhood relation and the cooling schedule [12, 13].

A new neighborhood distribution for SS+SA: FlipN. In [22] it was observed that for sparse graphs the best results with Local Search were obtained using 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.

Since the SA phase of SS+SA 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. Therefore, 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 1 supports this affirmation*: for each one of the $n(n-1)/2$ possible moves in the Flip2 neighborhood, we have computed their gain δ 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 φ obtained by SS. Clearly, the distributions follow a Gaussian bell and 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 (μ) and deviation (σ) of the distances between acceptable moves (the size of this sampling has been fixed to be $n\sqrt{n}$). Then, during the SA process, moves will be generated producing pairs of vertices whose distance follow a normal distribution $\mathcal{N}(\mu, \sigma)$. We call this new neighborhood **FlipN**.

*All the figures can be found in the Appendix. Many of them look better when seen in colors. They also can be found at http://www.lsi.upc.es/~jpetit/MinLA/SS+SA on the WWW.

Cooling schedule. The design of the cooling schedule for SS+SA is similar to the classical ones [12, 13, 22]. We use a geometric cooling schedule that decrements the temperature at each round multiplying it by α (usually $0.9 < \alpha < 1$). At each temperature, a round of r moves will be proposed. If these moves reduce the objective function more than a ratio ρ, we assume that this temperature has not yet reached equilibrium and so we perform another round at the same temperature. The SA ends when less than n 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, $t_0 = 10$ as starting temperature and $\rho = 0.9999$ for the equilibrium ratio.

(Alg. 1)

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

3 The Parallel SS+SA Heuristics

A general framework to parallelize SA on DMMs is presented in [5]. Some other papers have presented data level parallelizations for SA for other placement problems on shared memory machines (e.g. [2]). 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 that different processors evolve on the basis of a coherent state; *chaotic methods* admit that the gain computation involves some errors (that are expected to be small). In this section, we adapt these techniques to SS+SA for a DMM.

Exact parallel SS+SA. Given an integer p representing the number of available processors, a layout φ on a graph $G = (\{1, \ldots, n\}, E)$ 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 $\mathcal{V}_0$ be the p-partition induced by $j_i = in/p$ $(\forall i \in 1, \ldots, n-1)$, let $\mathcal{V}_{+1}$ be the p-partition induced by $j_i = in/p + n/2p$ and let $\mathcal{V}_{-1}$ 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 φ using SS. This step is done sequentially, because it can be done very fast. Afterwards, we sample in parallel the Flip2 neighborhood of φ to obtain the values of μ and σ. At this moment, the SA algorithm begins. The schedule is the same than in the sequential algorithm, but now the Metropolis process is different. The data level parallelization of Metropolis computes p-partitions of V using the current layout φ and concurrently applies moves within the FlipN neighborhood on these partitions, one partition per processor. The main procedure of the parallel SS+SA using p processors is:

function *Parallel SS+SA*(G) **is** *(Alg. 2)*
  Sequentially, generate an initial layout φ using Spectral Sequencing
  In parallel, sample the neighborhood at t_0 to obtain μ and σ
  $t := t_0$; $r := \beta \cdot \mu \cdot n/4p$
  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 $b < n$
  return $\langle \varphi, c_\varphi \rangle$
end

The *Metropolis(x)* 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 for an expensive communication process, without affecting the correctness of the algorithm. After r iterations, the algorithm perform a a synchronization between all the processors. 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:

function *Metropolis*(x) **is** *(Alg. 3)*
  for each processor $i \in \{1, \ldots, p\}$ **do in parallel**
    Get a private copy of layout φ
    From $\mathcal{V}_x$ compute V_i and compute the frontier nodes
    (* Perform Metropolis in $\mathcal{V}_x$ *)
    $b_i := 0$
    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 $\min\{1, e^{-\delta/t}\}$ **do**
          $\varphi := \varphi'$; **if** $\delta > 0$ **then** $b_i := b_i + 1$
        end with
      end if
    end repeat
  end for
  (* Synchronize and Gather data *)

Rebuild a global layout φ from the ones distributed among procs according to $\mathcal{V}_x$
$b := b + \sum_{i=1}^{p} b_i$
end

The basic reason for this parallel heuristic to work is that on sparse clustered graphs, partitions induced by a good layout have a few cutting edges. As a consequence, it is expected that 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 DMM 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 feasible layout. However, after the move, other processors would compute δ 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.

4 Experimental Evaluation

In this section we analyze and compare some empiric results aiming to evaluate and tune the performance of SS+SA. In order to help the reader to reproduce and verify the measurements mentioned in this research, its code, instances and raw data are available at http://www.lsi.upc.es/~jpetit/MinLA/SS+SA on the WWW. The experimental conditions are described in the appendix.

The benchmark graphs. 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 from the following families:

- A graph with known optimum: `mesh33` is a 33×33 mesh.
- Geometric random graphs $\mathcal{G}_{n,r_-,r_+}$: 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_+. For `geometric`, $n = 5000$, $r_- = 0.027$ and $r_+ = 0.04$.
- Thin geometric random graphs $\mathcal{G}_{\sqrt{n},p,r}$: 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 distance is at most r. For `thingeo`, $\sqrt{n} = 125$, $p = 0.33$ and $r = 4$.
- Random linear graphs (see Section 5): `rlg` belongs to $\mathcal{L}_n$ 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 SS heuristic, we applied it to our benchmarking graphs. Figure 2 compares SS with some other heuristics described in [22]. For all the graphs, the results show that SS is clearly the heuristic that provides better results. Moreover, it is considerably faster than the other candidates.

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 nodes that are close in the current layout with respect to flip the labels of any pair of nodes. For the particular case of the `airfoil1` graph, Figure 4 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 5. 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, the figures do not reflect that when using FlipN, a big part of the moves have a null gain ($\delta = 0$).

Improving SS solutions by SA. Is SA able to improve solutions generated by SA? Let C_{ss} be the cost of a solution computed by SS and let C_{sa} be the cost of a solution computed by SA starting with the SS solution. Then, the *improvement of* SA *over* SS is measured as $\frac{C_{\text{ss}} - C_{\text{sa}}}{C_{\text{ss}}}$. Table 3 shows the average improvements obtained for the sequential and parallel heuristics. For sequential SS+SA, one can observe that SA is always able to improve the SS solutions (usually more than a 16%). For exact parallel SS+SA, the improvement decreases as the number of processors increases. For chaotic parallel SS+SA, the improvement remains almost the same independently of the number of processors.

Quality and time of the parallel SS+SA heuristics. We have measured the quality of the solution obtained by our heuristics as well as the time required to compute them. The absolute measurements for sequential SS+SA are shown in Table 2. This table also reports that the standard deviation of measures is very low. In order to enable a comparison of the parallel heuristics with respect to the sequential one, 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. Table 4 shows for each graph the time efficiency and solution ratio obtained (in average). Figure 6 shows a trace of the annealing curves for the `airfoil1` and `thingeo` graph. From these results, it can be observed that:

- Time efficiency is always close to the 100% (except for `rlg`). 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 `rlg`). This fact shows that the parallel heuristics deliver solutions of comparable quality to the sequential one.

- In general, chaotic parallel SS+SA gives slightly better solutions than its exact version.
- 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 that `rlg` shows the limitations on the scalability of exact parallel SA; as shown in Section 5, this graph is a 2-expander and therefore almost all nodes are blocked when using more than 6 processors.

Visualization of the solutions. In order to have a "visualization" of the solutions delivered by SS+SA, Figure 8 shows `airfoil1` colored according to the cost of its nodes.

5 Random graphs with good estimations of their optima

In this section we present two classes of random graphs that are useful to generate MinLA instances for which a good estimate of the optimum is known by construction but is hidden from the heuristics (whp). Previously, the only known families of graphs that satisfied this property were deterministic [9, 18, 1, 25, 3, 19].

Let $\mathcal{L}_n$ be the class of *random linear graphs* on vertex set $V = \{1, \ldots, n\}$ where each possible edge $\{i, j\}$ appears with probability $1/|j - i + 1|$. It is easy to see that the expected number of edges in a $\mathcal{L}_n$ graph is concentrated around $\Theta(n \log n)$. The *canonical layout* ψ of a $\mathcal{L}_n$ is defined by $\psi(i) = i$ $(\forall i \in \{1, \ldots, n\})$. For almost all graphs in $\mathcal{L}_n$, we have that $C(\psi) = n^2/2 + \mathcal{O}(n \log n)$, whereas the expected cost of a random layout is $\Theta(n^2 \log n)$.

Let $\mathcal{H}_n$ be the class of *hidden line graphs* on vertex set $V = \{1, \ldots, n\}$ where each possible edge $\{i, j\}$ appears in the graph with probability 1 if $|i - j| = 1$ and probability $1/2^{|i-j|}$ otherwise. Essentially, hidden line graphs are line graphs with a few additional edges. For almost all graphs in $\mathcal{L}_n$, we have that $C(\psi) = \frac{5}{2}n - \frac{13}{2} + 62^{-n} + 2^{1-n}n$, whereas the expected cost of a random layout is $\Theta(n^2)$.

Theorem 1 For almost all random linear graphs and for almost all hidden lines graphs, the canonical layout is a constant approximation to its MinLA value.

Sketch of the proof. We say that a graph $G = (V, E)$ is c-expander if for all $U \subseteq V$ with $|U| \leq n/2$, it holds that $\beta(U) = |\{uv \in E \ : \ u \in U \wedge v \notin U\}| > c|U|$. One can prove that, almost all graphs in $\mathcal{L}_n$ are 2-expanders. Consider any layout φ of a 2-expander. Then, taking the family of subsets $U_i = \{1, \ldots, i\}$ for $i \in \{1, \ldots, n/2\}$ and $U_i = \{i, \ldots, n\}$ for $i \in \{n/2 + 1, \ldots, n\}$, we have that $C(\varphi) \geq \sum_{i=1}^{n} \beta(U_i) = 4(1 + 2 + \cdots + n/2) = \Theta(n^2)$.

On the other hand, applying the "degree bound" of [22] on $G \in \mathcal{H}_n$, one obtains that, for almost all hidden line graphs, $C(\varphi) \geq 2n + 42^{-n} - \frac{1}{3}4^{-n} + 2^{1-n}n - \frac{11}{3}$ for any layout φ. □

6 Hill Climbing can fail on line graphs

Let I_n be the *line graph* with vertex set $V = \{1, \ldots, n\}$ and edge set $\{\{i, i+1\}\}_{i=1}^{n-1}$. This graph has only two layouts with minimal cost: $\varphi_1^*(i) = i$ and $\varphi_2^*(i) = n - i + 1$ with $C(\varphi_1^*) = C(\varphi_2^*) = n - 1$. Moreover, the maximum cost value for a layout in I_n is $(n^2 + n)/2$ and the expected value of the cost of a randomly generated layout is $\Theta(n^2)$. In the following, we

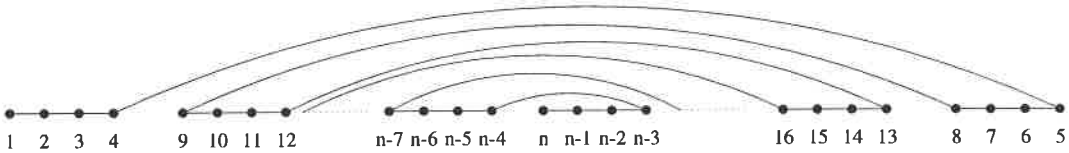
consider the application of Hill Climbing with the Flip2 neighborhood on a line graph. This algorithm on this class of graphs has a nice property:

Lemma 1 From any starting layout φ, 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:

Theorem 2 There is a layout π_s of I_n such that HC needs at least $\Theta(n^3)$ steps to hit an optimum. However, starting from π_s, the *expected* number of steps needed to hit an optimal solution is exponential.

Sketch of the Proof. Consider as π_s:



□

References

[1] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM J. Appl. Math.*, 25(3):403–423, November 1973.

[2] F. Darema, S. Kirkpatrick, and V. A. Norton. Parallel algorithms for chip placement by SA. *IBM Journal of Research and Development*, 31:391–402, May 1987.

[3] J. Díaz, A. Gibbons, G. Pantziou, M. Serna, P. Spirakis, and J. Torán. Parallel algorithms for the minimum cut and the minimum length tree layout problems. *Theoretical Computer Science*, (181):267–288, 1997.

[4] J. Díaz, J. Petit i Silvestre, M. Serna, and L. Trevisan. Approximating layout problems on random sparse graphs. Report de recerca, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 1998.

[5] R. Diekmann, R. Luelling, and J. Simon. A general purpose distributed implementation of SA. In *4th IEEE Symposium on Parallel and Distributed Processing*, 1992.

[6] G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-Conquer Approximation Algorithms via Spreading Metrics. In *36th FOCS*, pages 62–71, 1995.

[7] A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *37th FOCS*, pages 12–20. Los Alamitos, CA, 1996.

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

[9] L. H. Harper. Chassis layout and isoperimeter problems. *Preprint of Jet Propulsion Lab., California Institute of Pasadena*, 1972.

[10] B. Hendrickson and R. Leland. The Chaco User's Guide. 1995.

[11] M. Jerrum and G. Sorkin. Simulated Annealing for Graph Bisection. In *34th FOCS*, pages 94–103, 1993.

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

[13] 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.

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

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

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

[17] W. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of State Calculations by Fast computing machines. *J. Chem. Phys.*, (21):1087–1092, 1953.

[18] G. Mitchison and R. Durbin. Optimal numberings of an $n \times n$ array. *SIAM J. Alg. Disc. Math.*, 7(4):571–582, 1986.

[19] K. Nakano. Linear layouts of generalized hypercubes. In *Graph-theoretic concepts in computer science (Utrecht)*, volume 790 of *LNCS*, pages 364–375. Berlin, 1994.

[20] A. Nolte and R. Schrader. Coloring in sublinear time. In R. Burkard and G. Woeginger, editors, *Algorithms-ESA '97*, pages 388–401. LNCS, Springer-Verlag, 1997.

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

[22] J. Petit i Silvestre. Approximation Heuristics and Benchmarkings for the MinLA Problem. In R. Battiti and A. Bertossi, editors, *Alex '98 — Building bridges between theory and applications*, http://www.lsi.upc.es/~jpetit/MinLA/Benchmark, February 1998. Università di Trento.

[23] S. Rao and A. W. Richa. New Approximation Techniques for Some Ordering Problems. In *9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 211–218, 1998.

[24] G. Sasaki. The effect of the density of states on the Metropolis algorithm. *IPL*, 37(3):159–163, 1991.

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

[26] D.A. Spielman and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite elements meshes. In *37th FOCS*, 1996.

A Appendix: Summary of the Experimental Results

Name	Nodes	Edges	Degree	LB	Best
4elt	15606	45878	3/5.9/10	300675	2241776
airfoil1	4253	12289	3/5.78/10	24220	278253
crack	10240	30380	3/5.93/9	64938	1502544
geometric	5000	32983	2/13.2/28	135092	2194430
mesh33	1089	2112	2/3.87/4	*31680	*31680
rlg	2000	12448	3/12.44/27	1031249	1953731
thingeo	5167	39924	3/15.4/27	182589	1699145
whitaker3	9800	28989	3/5.91/8	57824	1169065

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 canonical arrangement of rlg is 1990793.

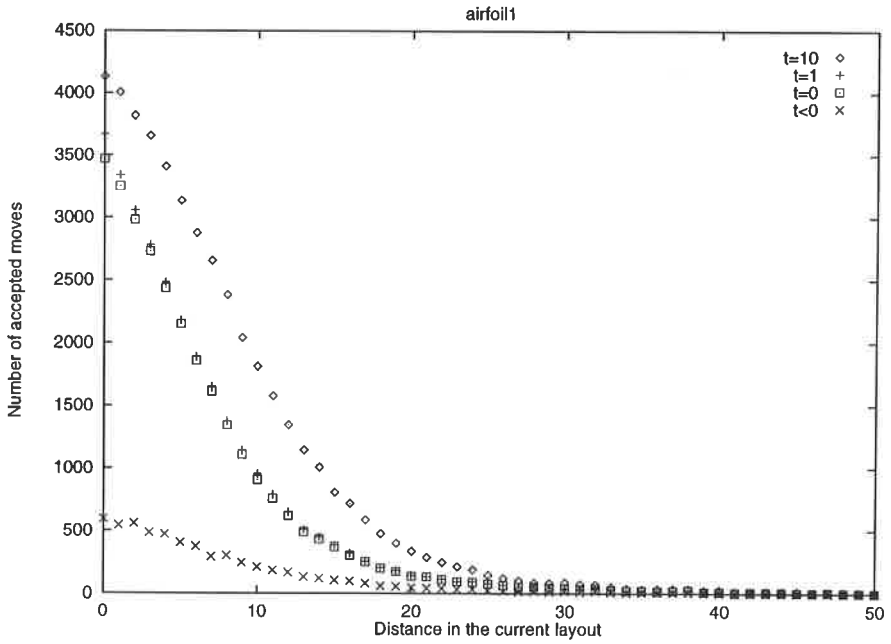


Figure 1: Number of accepted moves in function of their distance and the temperature (t) on the solution found by Spectral Sequencing for the airfoil1 graph. ($t < 0$ means that only strictly descendent moves are accepted.)

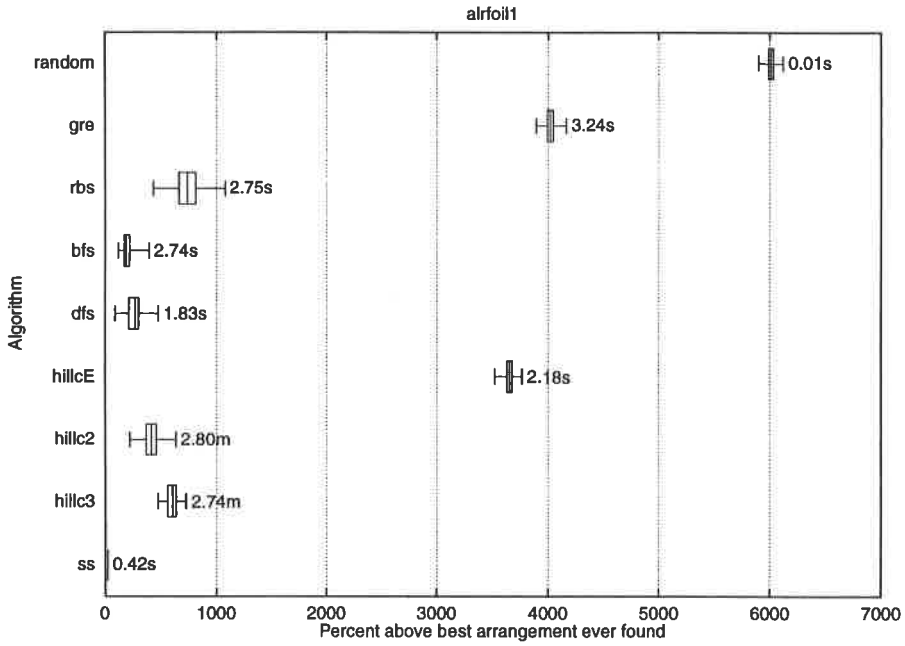


Figure 2: Comparison of previous heuristics on the `airfoil1` graph: random (**random**), variations of greedy successive augmentation algorithms (**gre**, **rbs**, **bfs dfs**), variations of HC (**hillcE**, **hillc2**, **hillc3**) and SS (**ss**). The boxplots represent the distribution of the approximations found in 1000 independent experiments and the average time needed.

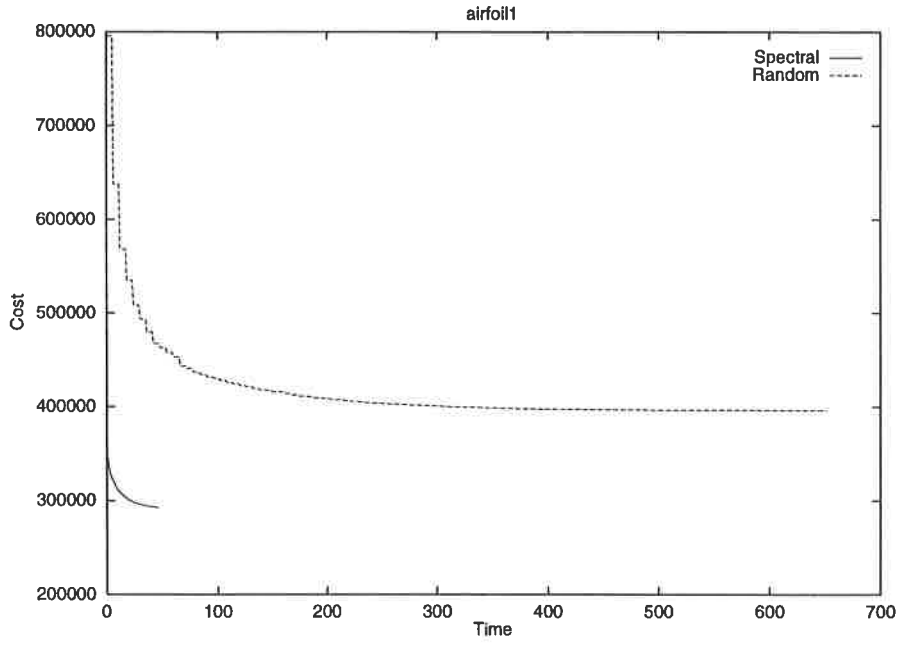


Figure 3: Old annealing curves on the `airfoil1` graph when using a random solution or the SS solution as initial solutions and the Flip2 neighborhood. Each unit of time corresponds to 2 minutes.

Graph	#Runs	Time (s)		Cost		
		Avg	%Dev	Avg	%Dev	Min
4elt	1	5187	—	2284690	—	2284687
airfoil1	20	710	6.33	286952	0.23	284727
crack	1	2480	—	1502700	—	1502696
geometric	10	1837	6.43	2210800	0.57	2194430
mesh33	20	164	8.11	33985	1.17	33233
rlg	10	1220	5.44	1953990	0.00	1953756
thingeo	10	856	4.73	1702890	0.17	1699145
whitaker3	10	2123	5.42	1187840	0.28	1183196

Table 2: Experimental results obtained with sequential SS+SA. Column #Runs shows the number of independent runs performed for each graph; column Time shows the total computation time (in seconds); column Cost shows the cost of the solution returned by SS+SA. Columns Avg show the average; columns %Dev give the percentual error in the standard deviation, that is, 100 · Dev/Avg; column Min gives the best cost obtained on all the runs.

Exact Parallel SS+SA

Processors	1	2	4	6	8	16
4elt	16.87	16.79	16.69	16.72	16.17	13.42
airfoil1	20.67	20.60	20.44	19.55	18.29	10.81
crack	8.30	8.30	8.21	8.17	8.05	7.10
geometric	17.74	17.83	16.94	14.76	10.49	1.63
mesh33	17.06	15.28	13.55	11.53	9.34	2.22
rlg	17.39	8.54	0.44			
thingeo	19.23	19.18	18.80	17.53	15.24	4.77
whitaker3	5.26	5.49	5.54	5.38	5.27	4.87

Chaotic Parallel SS+SA

Processors	1	2	4	6	8	16
4elt	16.87	16.94	16.87	17.29	17.32	16.88
airfoil1	20.67	20.70	20.61	20.76	20.68	20.65
crack	8.30	8.24	8.31	8.29	8.24	8.22
geometric	17.74	18.12	17.61	14.82	17.93	18.18
mesh33	17.06	16.43	16.70	16.58	16.65	16.14
rlg	17.39	17.39	17.39			
thingeo	19.23	19.26	19.26	19.25	19.25	19.14
whitaker3	5.26	5.46	5.44	5.53	5.39	5.24

Table 3: Average improvement (in %) obtained by SA on the solution found by SS depending on the number of processors and parallel strategy.

Exact Parallel SS+SA: Cost Ratio

Processors	1	2	4	6	8	16
4elt	100	99.91	99.79	99.83	99.17	96.02
airfoil1	100	99.91	99.71	98.61	97.09	88.95
crack	100	100.00	99.90	99.85	99.72	98.71
geometric	100	100.11	99.03	96.50	91.89	83.62
mesh33	100	97.90	95.94	93.75	91.48	84.83
rlg	100	90.33	82.98			
thingeo	100	99.93	99.46	97.94	95.29	84.81
whitaker3	100	100.25	100.30	100.13	100.02	99.59

Chaotic Parallel SS+SA: Cost Ratio

Processors	1	2	4	6	8	16
4elt	100	100.08	100.00	100.51	100.55	100.02
airfoil1	100	100.04	99.93	100.11	100.01	99.97
crack	100	99.93	100.01	99.98	99.92	99.91
geometric	100	100.45	99.84	96.57	100.22	100.53
mesh33	100	99.25	99.56	99.43	99.50	98.90
rlg	100	100.00	100.00	100.00	100.00	100.00
thingeo	100	100.03	100.03	100.02	100.02	99.89
whitaker3	100	100.22	100.20	100.29	100.14	99.99

Exact Parallel SS+SA: Time Efficiency

Processors	1	2	4	6	8
4elt	100	102.71	99.22	112.10	110.49
airfoil1	100	96.39	98.26	99.06	98.63
crack	100	93.73	98.37	97.60	97.55
geometric	100	101.06	105.76	113.60	126.79
mesh33	100	101.94	105.13	109.75	120.12
rlg	100	198.81	4523.02		
thingeo	100	98.72	101.36	108.54	110.93
whitaker3	100	98.69	100.92	103.22	103.43

Chaotic Parallel SS+SA: Time Efficiency

Processors	1	2	4	6	8
4elt	100	104.83	107.26	94.87	103.39
airfoil1	100	97.23	97.54	96.39	94.71
crack	100	98.96	93.87	98.11	100.49
geometric	100	94.80	100.64	98.38	98.33
mesh33	100	99.86	97.02	95.57	91.69
rlg	100	100.43	98.70	102.08	100.71
thingeo	100	96.34	97.04	96.89	94.00
whitaker3	100	98.54	101.19	97.57	96.95

Table 4: Average solution ratio and time efficiency (both in %) for exact and chaotic parallel SS+SA relatives to sequential SS+SA. Time measures for 16 processors are not given because they were not taken in exclusive mode (virtual processors were used).

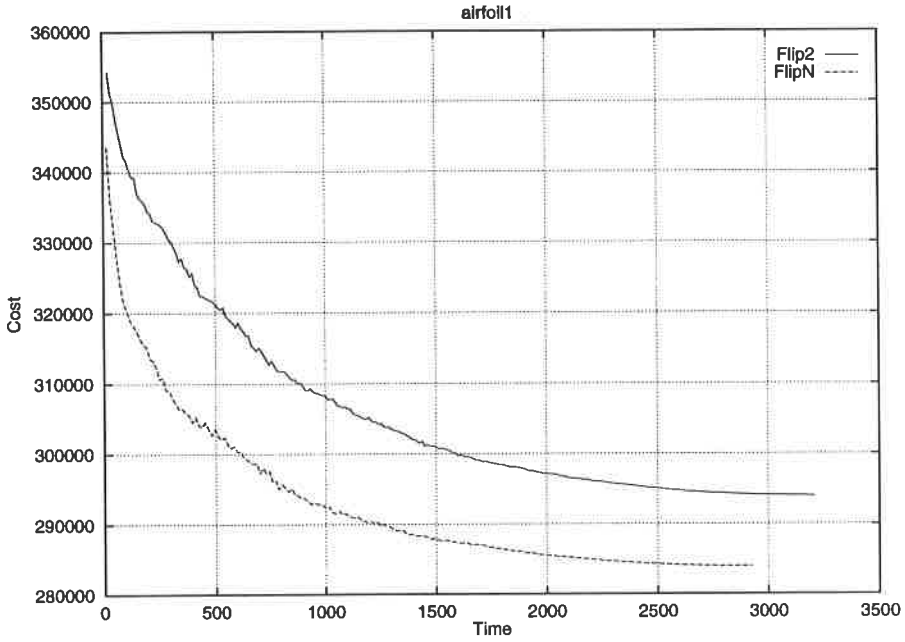


Figure 4: SS+SA on `airfoil1` using the Flip2 or FlipN neighborhoods: Evolution of cost in function of time (in seconds).

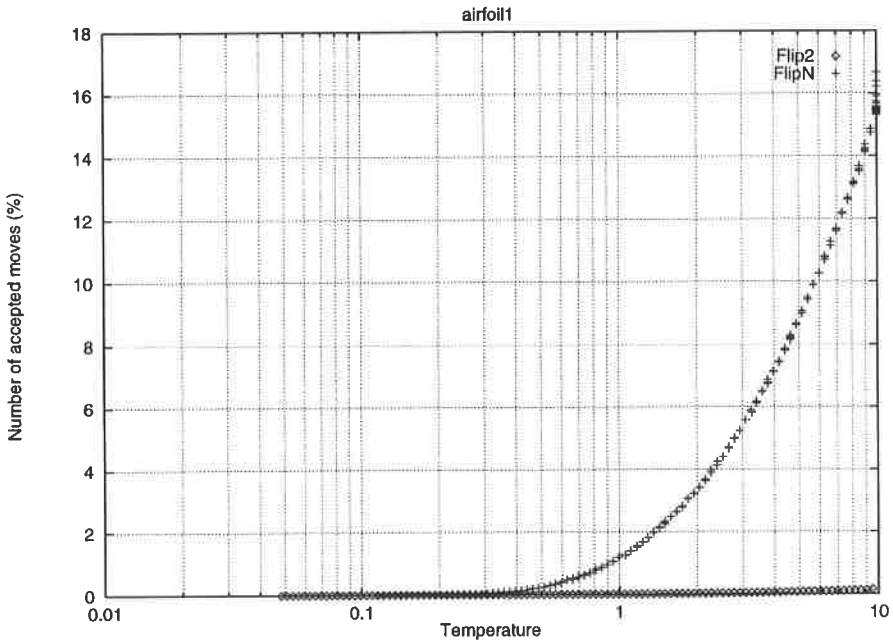


Figure 5: SS+SA on `airfoil1` using the Flip2 or FlipN neighborhoods: Percent of accepted moves in function of temperature (the time goes right to left).

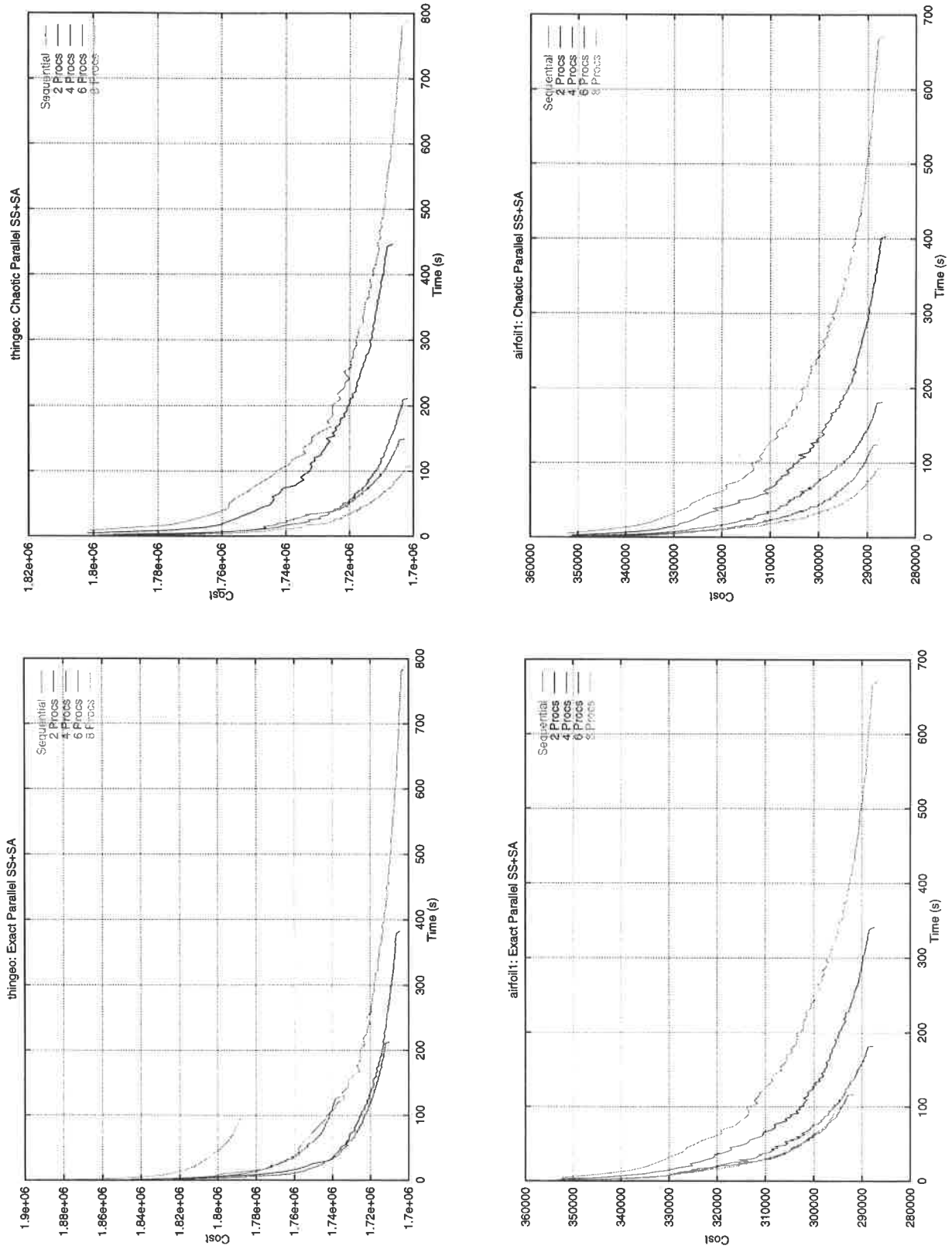
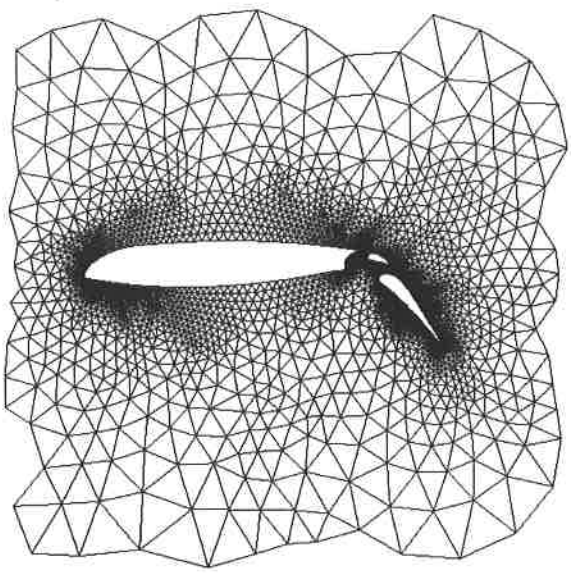
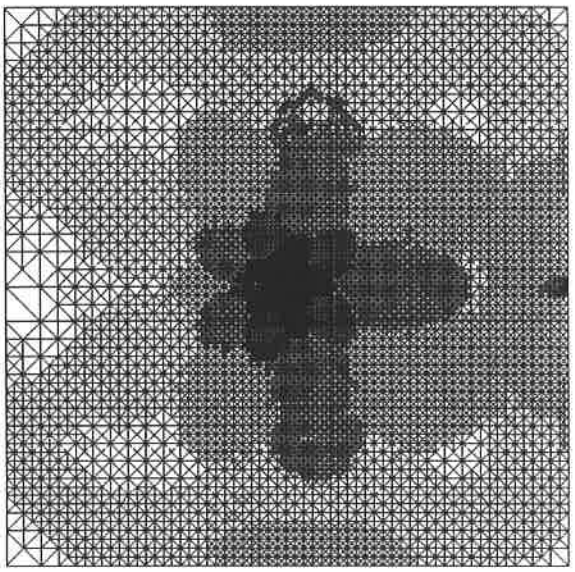


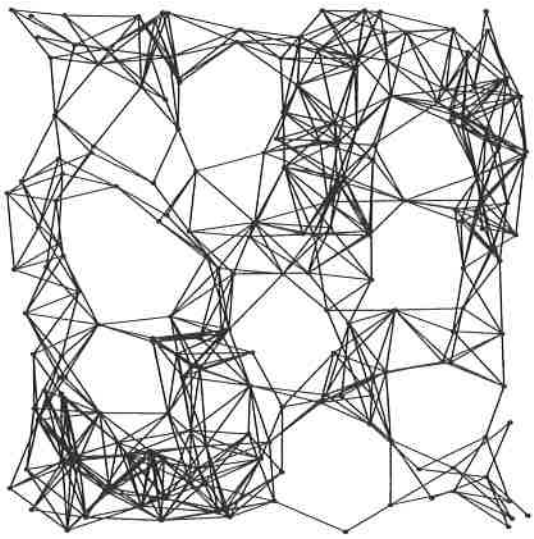
Figure 6: Exact and Chaotic Parallel SS+SA on `thingeo` and `airfoil1`.



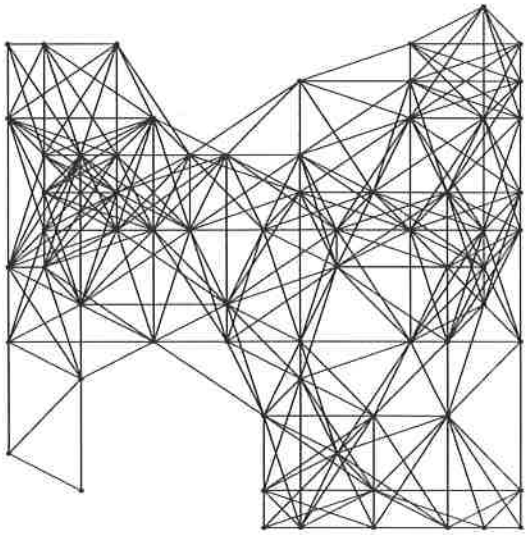
(a) **airfoil1**



(b) **crack**



(c) A random geometric graph

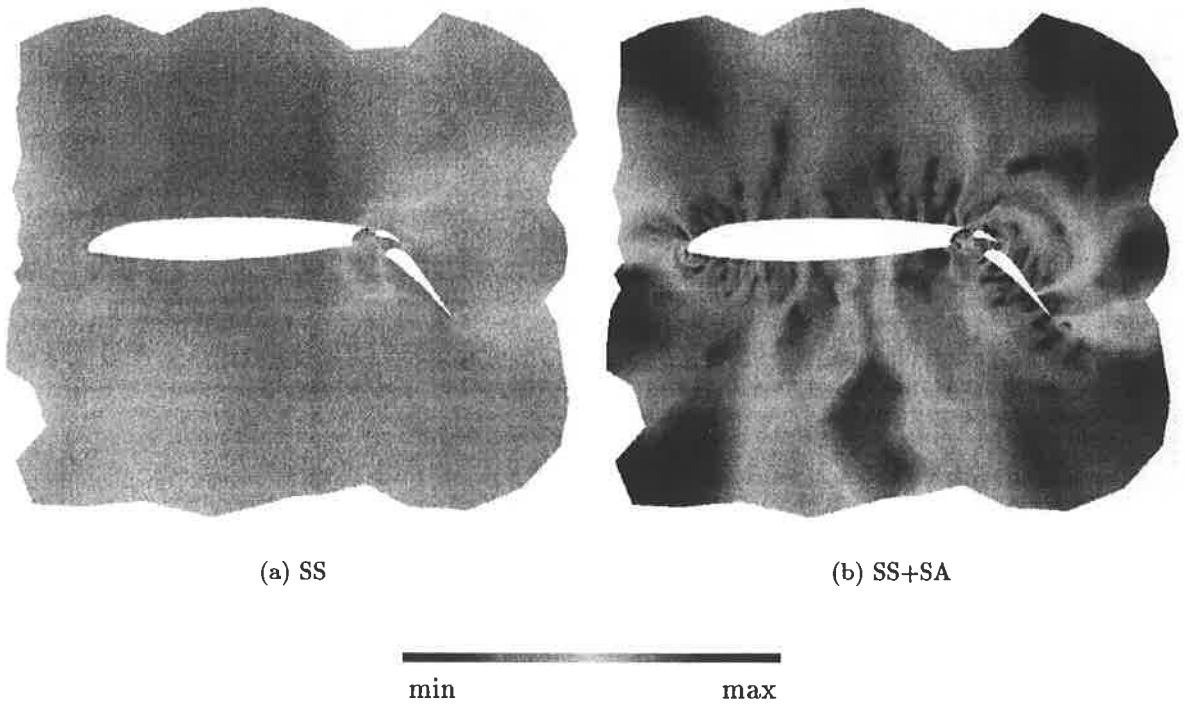


(d) A random thin graph

Figure 7: Some of the benchmarking graphs (In fact, **geometric** and **thingeom** are bigger than the shown ones).

Experimental conditions. The sequential SS+SA heuristic has been coded in the C++ programming language. The parallel heuristics use C++ augmented with MPI. All the programs have been run on an IBM SP2 computer with 8 Power2 processors under AIX. 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. 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].



(a) SS (b) SS+SA

Figure 8: Visualization of the MINLA solutions on `airfoil1`. (a) shows the solution found by SS and (b) shows the solution found by SS+SA. The visualization of a solution is done by coloring each node of the graph according to its contribution to the total cost (see the spectrum) together with a shadowing post-process. One can observe that SS generates a quite regular solution (all nodes have almost the same contribution to the total cost). The figure also seems to show that SS+SA improves SS by improving certain local zones of the graph with high cost nodes separating them. Note: this figure is best seen printed in colors or using a Postscript previewer as `ghostview`. It can also be found at http://www.lsi.upc.es/~jpetit/MinLA/SS+SA on the WWW.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1998

LSI–98–1–R "Optimal Sampling Strategies in Quicksort and Quickselect", Conrado Martinez, Salvador Roura.

LSI–98–2–R "Query, PACS and simple-PAC Learning", J. Castro and D. Guijarro.

LSI–98–3–R "Interval Analysis Applied to Constraint Feasibility in Geometric Constraint Solving", R. Joan-Arinyo and N. Mata.

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

LSI–98–5–R "Some reflections on applying Workflow Technology to Software Process", Camilo Ocampo and Pere Botella.

LSI–98–6–R "Surface Fairing for Ship Hull Design Application", P. Brunet, A. Vinacua, M. Vivó, N. Pla, A. Rodríguez.

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. Àlvarez, J. Diaz and M. Serna.

LSI–98–10–R "A unified approach to natural language treatment", Jordi Àlvarez.

LSI–98–11–R "Collision Detection: Models and Algorithms", Marta Franquesa and Pere Brunet.

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

LSI–98–13–R "HyperChromatic trees: a fine-grained approach to distributed algorithms on RedBlack trees", Xavier Messeguer and Borja Valles.

LSI–98–14–R "Asynchronous Interface Specification, Analysis and Synthesis", Michael Kishinevsky, Jordi Cortadella, Alex Kondratyev and Luciano Lavagno.

LSI–98–15–R "Heuristics for the MinLA Problem: Some Theoretical and Empirical Considerations", Josep Diaz, Jordi Petit i Silvestre and Paul Spirakis.

LSI–98–16–R "Sampling matchings in parallel", Josep Diaz, J. Petit i Silvestre, María Jose Serna and Paul Spirakis.

LSI–98–17–R "The Parallel Approximability of the False and True Gates Problems for Nor Circuits", M. Serna and F. Xhafa.

LSI–98–18–R "Basic Geometric Operations in Ruler-and-Compass Constraint Solvers using Interval Arithmetic", R. Joan-Arinyo and N. Mata.

LSI–98–19–R "HDM: AN HETEROGENEOUS STRUCTURES DEFORMATION MODEL", Montse Bigordà and Dani Tost.

LSI–98–20–R "Visualization of Cerebral Blood Vessels", Anna Puig.

LSI–98–21–R "Cerebral Blood Vessels Modelling", Anna Puig.

LSI–98–22–R "Discrete Medial Axis Transform for Discrete Objects", Anna Puig.

LSI–98–23–R "Incorporating the Behavioural Information to the Schema Construction Processs of Federated Data Bases System", Luis Carlos Rodríguez G.

LSI–98–24–R "Del Texto a la Información", J. Atserias, N. Castell, N. Català, H. Rodríguez and J.Turmo.

LSI–98–25–R " Construcción automática de diccionarios de patrones de extracción de información", Neus Català and Núria Castell.

LSI–98–26–R "Syntactic Connectivity", Glyn Morrill.

LSI–98–27–R "Geometric Distance Constraint Satisfaction by Constraint-to-constraint relaxation", Lluis Solano Albajes and Pere Brunet Crosa.

LSI–98–28–R "Modelling Surfaces from Planar Irregular Meshes", Josep Cotrina Navau and Nuria Pla Garcia.

LSI–98–29–R "Computing Directional Constrained Delaunay Triangulations", Marc Vigo Anglada .

LSI–98–30–R "On the complexity of moving vertices in a graph", Antoni Lozano (Universitat Politècnica de Catalunya) and Vijay Raghavan (Vanderbilt University).

LSI–98–31–R "Defining and Translating Visual Schemas for Deductive Databases", Jordi Puigsegur, Joan A. Pastor, and Jaume Agustí.

LSI–98–32–R "A framework for Animation in Global Illumination Environments", I. Martin, X. Pueyo and D. Tost.

LSI–98–33–R "Fuzzy Heterogeneous Neurons for Imprecise Classification Problems", Julio J. Valdés, Lluís A. Belanche, René Alquézar.

LSI–98–34–R "Query Containment Checking as a View Updating Problem", C. Farré, E. Teniente and T. Urpí.

LSI–98–35–R "A Computational Characterization of Collective Chaos", Jordi Delgado and Ricard V. Solé.

LSI–98–36–R "Lexicographic product and Boustrofedonic product in Unrank", Xavier Molinero Albareda.

LSI–98–37–R "Fringe analysis of synchronized parallel algorithms on 2–3 trees", R. Baeza-Yates, J. Gabarró and X. Messeguer.

LSI–98–38–R "Invocació explícita versus invocació implícita: Anàlisi comparativa de dos enfocaments de disseny de Sistemes d'Informació", Cristina Gómez, Luis Felipe Fernández, Juan Ramón López and Antoni Olivé.

LSI–98–39–R "Practical Algorithms for On-line Sampling", Carlos Domingo, Ricard Gavaldà and Osamu Watanabe.

LSI–98–40–R "A Geometric Relaxation Solver for Parametric Constraint-Based Models", Lluis Solano Albajes and Pere Brunet Crosa .

LSI–98–41–R "A role of constraint in self-organization", Carlos Domingo, Ricard Gavaldà and Osamu Watanabe.

LSI–98–42–R "Heuristics for the MinLA Problem: An Empirical and Theoretical Analysis", Josep Díaz, Jordi Petit i Silvestre, María José Serna and Paul Spirakis.

LSI–98–43–R "Efficient Read-Restricted Monotone CNF/DNF Dualization by Learning with Membership Queries", Carlos Domingo, Nina Mishra and Leonard Pitt.

Hardcopies of reports can be ordered from:

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

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