Alcom-FT Technical Report Series
ALCOMFT-TR-03-83

Parallel Processing Letters,

COMBINING SPECTRAL SEQUENCING AND PARALLEL SIMULATED ANNEALING FOR THE MINLA PROBLEM *

JORDI PETIT

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya
Campus Nord C6. Jordi Girona 1–3, 08034 Barcelona, Catalonia
jpetit@lsi.upc.es

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

In this paper we present and analyze new sequential and parallel heuristics to approximate the Minimum Linear Arrangement problem (MinLA). The heuristics consist in obtaining a first global solution using Spectral Sequencing and improving it locally through Simulated Annealing. In order to accelerate the annealing process, we present a special neighborhood distribution that tends to favor moves with high probability to be accepted. We show how to make use of this neighborhood to parallelize the Metropolis stage on distributed memory machines by mapping partitions of the input graph to processors and performing moves concurrently. The paper reports the results obtained with this new heuristic when applied to a set of large graphs, including graphs arising from finite elements methods and graphs arising from VLSI applications. Compared to other heuristics, the measurements obtained show that the new heuristic improves the solution quality, decreases the running time and offers an excellent speedup when ran on a commodity network made of nine personal computers.

1. Introduction

Several well-known optimization problems on graphs can be formulated as *layout problems*. Their goal is to find a layout (ordering) of the vertices of an input graph such that a certain function is minimized. Layout problems are an important class of problems with many different applications in Computer Science, VLSI design, Biology, Archaeology, Linear Algebra, etc; see [1] for a survey on layout problems. Finding an optimal layout is **NP**-hard in general, and therefore it is natural to develop efficient heuristics that give good approximations.

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

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

*This research was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and by the Spanish CICYT projects TIC2002-04498-C05-03 (TRACER) and TIC2000-1970-CE.

There exist fully polynomial time approximation schemes for MinLA on dense graphs [2], but not on sparse graphs. To date, the best polynomial time approximation algorithm for MinLA gives a $O(\log n)$ approximation factor for general graphs [3]. However, this algorithm presents the disadvantage of having to solve a linear program with an exponential number of constraints using the Ellipsoid method, and thus does not seem a practical option for large graphs. Therefore, it is reasonable to investigate heuristic methods that return good solutions in a moderated amount of time, and to speed them up by parallel processing.

In a previous work [4], the author presented a number of heuristics to approximate MinLA, and measured their behavior on a test suite of graphs. For the bigger graphs, the results were twofold: On one hand, the best results are obtained with Simulated Annealing (SA), which consumes an inordinate amount of time. On the other hand, results obtained with Spectral Sequencing (SS) are not much larger than the ones obtained by SA, and these are computed much faster.

In this paper, we address the problem of obtaining faster best solutions. To do so, in Section 2 we review the SS and SA heuristics. Then, in Section 3, we present a new sequential heuristic called SS+SA based on the combination of SS and SA. The basic idea of this new heuristic consists in building a globally good layout using SS and improving it locally through SA. Afterwards, in Section 4, we give two strategies to parallelize the SS+SA heuristic. Section 5 shows that one of these strategies preserves the quality of the obtained solutions and provides excellent speedups when run on a commodity network of personal computers. The paper is closed in Section 6 with a summary of our conclusions.

2. Review of SS and SA

Before presenting the new SS+SA heuristic, we review SS and SA.

The Spectral Sequencing heuristic to find layouts for MinLA is due to Juvan and Mohar [5]. Given a graph G, the heuristic first computes the *Fiedler vector* of G; that is, the eigenvector $x^{(2)}$ corresponding to the second smallest eigenvalue λ_2 of the Laplacian matrix L_G of G. Then, each position of $x^{(2)}$ is ranked. Thus, the spectral sequencing heuristic returns a layout φ satisfying

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

The rationale behind this heuristic is that the ordering of the vertices produced by their values in the Fiedler vector has some nice properties. In particular, vertices connected by an edge will tend to be assigned numbers that are close to each other. Moreover, this heuristic solves a closely related continuous problem [6].

The Simulated Annealing heuristic was proposed by Kirkpatrick *et al.* [7] and belongs to the class of randomized local search heuristics. The basic principle of local search is to iteratively improve a given solution by performing local changes on its combinatorial structure. In Hillclimbing, changes that improve the solution are accepted, whereas that changes that worse the solution are rejected. Hillclimbing terminates in a local optimum. In order to enable local search to escape from these local optima and accept downhill moves, Metropolis *et al.* [8] proposed an algorithm parametrized by a temperature t. A move that produces a gain of δ in the cost is accepted with probability $\min\left(1, e^{-\delta/t}\right)$. 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) [7]. The following algorithm applied to MinLA illustrates the general SA scheme:

function *SimulatedAnnealing*(G) :
  $\varphi :=$ Generate an initial layout
  $t := t_0 :=$ Select initial temperature
  while $\neg$frozen :
    while $\neg$equilibrium :
      $\varphi' :=$ Select a neighbor of φ
      $\delta := \text{LA}(\varphi, G) - \text{LA}(\varphi', G)$
      with probability $\min(1, e^{-\delta/t}) : \varphi := \varphi'$
    $t := \alpha t$
  return φ

Besides the selection of its parameters (such as α, frozen and equilibrium detection...), a key decision in SA is the election of a neighborhood, which corresponds to specify which are the local changes applicable to the current solution. In [4] the analysis of three different neighborhoods on sparse graphs showed that 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. An advantage of this neighborhood is the easiness to perform movements and the low effort necessary to compute incrementally $\text{LA}(\varphi', G)$ from $\text{LA}(\varphi, G)$.

3. The sequential SS+SA heuristic

Since SS is a constructive heuristic that builds a layout from scratch, while SA is a local search heuristic that improves a given layout, it makes sense to try to combine both methods in order to obtain a new heuristic to find better approximations in a faster time. We call SS+SA the new heuristic we propose. Its basic idea is to first build a "globally good layout" by the means of SS, and then improve it locally through the use of SA. This approach has several consequences.

First of all, recall that when using SA with random solutions, cooling schedules start with high temperatures that accept, say, one half of the generated movements. Of course, this behavior is not suitable for the SS+SA heuristic, as this would completely destroy the solution generated by SS. As a consequence, it will be necessary to start SA at a low temperature.

Since the SA process will be started with a quite good solution at a low temperature, it could be expected to have a high number of rejected moves in the Flip2 neighborhood. As a consequence, finding acceptable moves will be a difficult and long task that we wish to speed up. In order to reduce the time of this search, we can make use of the following idea: On a good solution, changes that are worth to be tried must be close in the current layout. It does not make sense to try to flip vertices that are far away! Figure 1 supports this heuristic 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 on a fixed graph. The figure shows how many moves have been accepted in function of the distance between the vertices in the layout obtained by SS. Here, the distance between two vertices u and v in a layout φ is defined as $|\varphi(u) - \varphi(v)|$. The distributions clearly follow a half Gauss bell, and show that moves involving close vertices in the layout will have higher probability of being accepted.

This observation gives rise to the use of a new neighborhood relation: Just after obtaining the solution found by SS, we perform a scanning of the Flip2 neighborhood and compute the mean μ and standard deviation σ of the distances between acceptable moves. 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. Generating moves and computing their gain in the FlipN neighborhood can be implemented efficiently.

The design of our cooling schedule is pragmatic, close to the classical ones and, in a big extend, influenced by our experience and the requirements on the running time and quality solution of SS+SA. We use a geometric cooling schedule that starts at an initial temperature t_0 and, at each round, decrements it by a factor of α. For each temperature, a Metropolis round of r moves will be generated. The SA process ends when the temperature drops below t_f. The concrete values of the parameters we have used are $t_0 = 10$, $t_f = 0.2$, $\alpha = 0.95$ and $r = 20n^{3/2}$.

The pseudo-code of the sequential SS+SA heuristic is as follows:

function *SS+SA*(G) :
  Generate an initial layout φ using SS
  Scan the neighborhood at t_0 to obtain μ and σ
  $t := t_0;\ n := |V(G)|;\ r := \beta n^{3/2}$
  while $t > t_f$:
    repeat r **times** :
      Select u and v with $|\varphi(u) - \varphi(v)|$ drawn from $\mathcal{N}(\mu, \sigma)$
      $\delta :=$ *GainWhenFlip2*(G, φ, u, v)
      with probability $\min(1, e^{-\delta/t})$: *Flip2*(φ, u, v)
    $t := \alpha t$
  return φ

4. Parallel strategies for SS+SA

In the following, we present two different strategies to parallelize the Metropolis loop of SS+SA. Following [9], we call them *exact* and *chaotic* strategies, in analogy to their shared memory algorithms.

We need first some definitions: Given an integer P, a layout φ on a graph $G = (V, E)$ with n vertices, and an increasing sequence of indices $j_0, j_1, \ldots, j_P$ with $j_0 = 0$ and $j_P = n$, let us define a P-partition $(V_1, V_2, \ldots, V_P)$ of V by

$$V_i = \{u \in V \mid j_{i-1} < \varphi(u) \leqslant j_i\}, \qquad \forall i \in [n].$$

Moreover, let $\mathcal{V}_0$ be the P-partition induced by $j_i = i \cdot n/P$, let $\mathcal{V}_{+1}$ be the P-partition induced by $j_i = i \cdot n/P + n/2P$, and let $\mathcal{V}_{-1}$ be the P-partition induced by $j_i = i \cdot n/2 - n/2P$.

Given a P-partition, edges whose vertices are in different partitions are said to be in the *cut*. Vertices that have an adjacent edge in the cut are said to be in the *frontier*; the remaining vertices are said to be *free*. See Figure 2 for an example. Notice that these three partitions group vertices that are consecutive in the layout.

Exact strategy. The *exact strategy* for the parallel SS+SA heuristic on P processors starts computing sequentially an initial layout φ using SS. Afterwards, the Flip2 neighborhood is scanned in parallel by the P processors, to obtain the values of μ and σ. At this moment, the SA algorithm begins. The cooling schedule is the same as in the sequential algorithm, but now the Metropolis process is different. The exact strategy for SS+SA using P processors is as follows:

function *ExactParallel SS+SA*(G) :
    Sequentially, generate an initial layout φ using Spectral Sequencing
    In parallel, sample the neighborhood at t_0 to obtain μ and σ
    $t := t_0;\ n := |V(G)|;\ r := \beta n^{3/2}/4P$
    while $t > t_l$ **do** :
        repeat r **times** :
            Metropolis(0); *Metropolis*$(+1)$; *Metropolis*(0); *Metropolis*(-1)
        $t := \alpha t$
    return φ

The *Metropolis* function is the one that contains parallelism. Given $G = (V, E)$, the basic idea is to compute P-partitions of V using the current layout φ, assigning one partition to each processor. Then, the processors concurrently generate moves in the FlipN neighborhood on their own partitions. If some of the vertices that a move proposed lies in the frontier, we say that the move is forbidden and we reject it. Otherwise, we accept it according to the Metropolis criterium. Forbidding moves is necessary in order to ensure that, during all this phase, the information owned by each processor is maintained coherent with respect to the information stored in the other processors. Notice that allowed moves do not need to be communicated to the remaining of processors, since they will never use this information. As a consequence, forbidding moves removes the need of an expensive communication while not affecting the correctness of the algorithm. After performing r iterations, a synchronization phase between all the processors is performed. During this synchronization, a global layout is rebuilt through the combination of the layouts in each processor and the P-partition. Afterwards, the same process is repeated, with a different P-partition.

The corresponding pseudo-code is as follows:

function *Metropolis*(x) :
    for each processor $i \in [P]$ **do in parallel** :
        Get a private copy of layout φ
        From $\mathcal{V}_x$ compute V_i and compute the frontier vertices
        repeat r **times** :
            Select u, v in V_i with $|\varphi(u) - \varphi(v)|$ drawn from $\mathcal{N}(\mu, \sigma)$
            if u, v are free w.r.t. V_i :
                $\delta :=$ *GainWhenFlip2*(G, φ, u, v)
                with probability $\min(1, e^{-\delta/t})$: *Flip2*(φ, u, v)
    Rebuild a global layout φ from the ones distributed
        among processsors, according to $\mathcal{V}_x$

On sparse graphs, partitions induced by a good layout are them-selves expected to 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 should be many opportunities to optimize the individual partitions. The trick of using three different partitions intercalated in four phases is used in order to not always forbid the same moves. Otherwise we would obtain layouts which would be well arranged inside each partition, but locally bad arranged at the frontiers.

Chaotic strategy. It is clear that forbidding moves enables the processors to allways have an up-to-date information. Unfortunately, these forbidden moves restrict the possibilities of optimizing, as they are directly rejected. If we admit that processors freely move vertices on the frontier, we get the *chaotic strategy* for the SS+SA heuristic. In this case, as moves are applied into a concrete partition but cannot cross the partitions, the global state of the system will still be coherent, since it represents a valid layout (bijection). However, after moving a fronterier vertex, other processors would compute the gains of their moves with an out-of-date information, thus commiting an error. We expect that this error should not be very large, and that it decreases as the temperature is lowered.

5. Experimental evaluation

We now present, compare and analyze some empirical results aiming to evaluate the performance of the SS+SA heuristic. We are interested in the relative performance and efficiency of SS+SA compared to SS and SA, and in the scalability in time and in quality between the different parallel strategies. Our results are based on a test suite of graphs proposed by the author in [4] and used in other experimental studies for MinLA [10]. We only consider here a subset of that test suite, discarding the small graphs and the random graphs.[†] The main characteristics of the graphs in the current test suite are shown in Table 1.

Experimental conditions. The sequential SS+SA heuristic has been coded in C++ and the parallel strategies use MPI. The programs have been compiled using GCC with maximum optimization compiler options and linked against LAM. The programs have been run on a Linux cluster made of nine identical personal computers with AMD K6 processors at 450 MHz and 256 Mb of memory with a Linux operating system. The PCs are connected with an inexpensive Fast Ethernet network at 100Mb/s. All the experiments have been executed in dedicated mode (expect for system daemons) and measure the total elapsed (wall-clock) time, starting at the moment that the input graph is read.

Comparing FlipN with Flip2. The following experiments were designed to evaluate the effect of using the FlipN neighorhood instead of the more traditional Flip2 neighorhood, i.e. to favor moves among close vertices in the current layout with respect to try to move arbitrary pairs of vertices. For the particular case of the `airfoil1` graph, Figure 3 shows a trace of the cost as function of the time when

[†]Small graphs are discarded because parallel processing is not suitable for them. Random graphs are discarded because the results in [11] show these graphs are not usefull to distinguish good from bad heuristics.

using Flip2 and FlipN. From the curves, the benefits of stretching the neighborhood in order to reduce the runtime and to increase the quality of the solution are evident.

Evaluation of the sequential SS+SA heuristic. In order to compare SS+SA against SS and SA alone, we have tried to apply SS+SA to the graphs in our test suite. Table 2 compares the solution quality and running time of SS+SA with the ones of SS and the better ones of SA. For the FE graphs, it can be observed that SS+SA improves the SS and SA solutions by more than 20%, while reducing the running time to a 25% of SA. For the remaining graphs, SS+SA allways improves the SS solutions and only for the `c5y` and `gd96a` graphs it is unable to improve the SA solution. The running times are usually lower for SS+SA than for SA. Overall, these results show that the SS+SA heuristic is a valuable improvement over SS and SA alone. In the case of the larger graphs (the FE family) the improvements are substantial, both in quality and time.

Evaluation of the parallel SS+SA heuristics. In order to compare the behavior of the exact and chaotic strategies for parallel SS+SA, we have measured the running time and solution quality of these heuristics on our graphs. The assesment of their goodness is done by comparison with the sequential heuristic: Let T the time needed to execute the sequential SS+SA heuristic and C the cost obtained, let T_P be the time of the parallel SS+SA heuristic ran on P processors and let C_P the cost obtained. The *speedup* on P processors is T/T_P, and the *efficiency factor* is $T/T_P P$. In order to compare the quality of the heuristics, we define the *quality factor* as C/C_P. Good parallelitzations ought to have the efficiency and quality factors close to one. In principle, the efficiency factor of any algorithm can never be larger than one, but due to the random nature of our heuristics, this is possible in our case.

The measured efficiency factor and quality factor for exact SS+SA are shown in Table 3.The results for chaotic SS+SA are shown in Table 4. Both tables are shown graphically in Figures 4 and 5. Traces of the parallel annealings curves for the `airfoil1` graph are shown in Figure 6.

From these measures, the following facts can be observed for the exact SS+SA strategy:

- For the VLSI graphs, exact parallel SS+SA does not work at all. The reason is that the number of forbidden moves is too high: A closer inspection reveals that in all these graphs, there is a vertex connected with more than 300 neighbors. This forbids too many moves as soon as $P \geqslant 3$.
- The quality factor of exact parallel SS+SA decreases as the number of processors increases. The trend shows that the quality factor depends on the size of the graph: For a fixed number of processors, larger graphs have a better quality factor.
- For the medium sized graphs, the efficiency factor of exact parallel SS+SA decreases as the number of processors increases. This does not seem to be the case for the larger graphs, for which the efficiency factor is close to 1.

Globally, these observations show that the exact strategy for SS+SA is not scalable and heavily depends on the number of forbidden moves. In the case of the chaotic strategy for parallel SS+SA, the following observations can be made:

- Chaotic SS+SA can process all the inputs, because no movements are forbidden.
- The quality factor of chaotic parallel SS+SA is always close to 1 for the FE graphs. Specifically, its range is between 0.99 and 1.01. As a consequence, the cost of the solutions obtained by chaotic parallel SS+SA and the cost of the solutions obtained by sequential SS+SA differ at most by 2%. This variability is the same than the one found for independent runs of sequential SS+SA.
- The efficiency factor of exact parallel SS+SA for medium sized graphs slightly decreases as the number of processors increases. The worst speedup is achieved with the `mesh33x33` graph with 9 processors: its efficiency factor is 0.779, which on a network of personal computer still looks good. For the larger graphs, the efficiency factor is usually grater than 95%.

From the previous observations the following facts can be inferred: With regards to solution quality, chaotic parallel SS+SA returns solutions of the same quality than sequential SS+SA. This is not true for exact SS+SA, for which the solution quality degrades as the number of processors increases. This shows, maybe surprisingly, that it is usefull to let the processors act without a complete knowledge of the state of the whole layout. With regards to the time efficiency, the results of chaotic SS+SA on 9 processors present an excelent speedup for the larger graphs and a good speedup for the medium sized graphs.

6. Conclusions

In this paper we have presented a new heuristic for the MinLA problem that combines SS with SA. The aim was to accelerate the SA heuristic and to obtain better solutions. For others problems, it has previously been pointed that spectral methods offer solutions that have a good global quality but combined with a local weakness, and that the global strength of SS combined with local search can lead to heuristics significantly better than either alone [12]. Our study gives further confirmation in that direction.

Designing the SS+SA heuristic, we have addressed three main points in the improvement of SA techniques: the use of better-than-random initial solutions, the use of an adequate neighborhood specially adapted to improve good solutions, and the use of parallelism.

The use of initial solutions computed by SS has proved to be very valuable, as it provides a simple way to enable SA to reach good solutions in shorter time, specially in large graphs. Moreover, we have seen that SA is capable to further improve these solutions by factors around 20%.

The use of better-than-random solutions to start SA directly implies the use of a low starting temperature; but more subtely, it also gives rise to the use of more refined neighborhood relations that have a great influence on the runtime and the quality of the obtained solution. In the particular case of the MinLA problem, we have shown how to dynamically choose a neighborhood distribution depending on the input. This has been possible due to the sparsity of our graph instances and the good behaviour of the SS heuristic.

Regarding the parallelization of SA, we have proposed a tailored parallelization that enables different processors to perform Metropolis concurrently on different partitions of the graph. In the exact strategy, the global state is always coherent, in the sense that it represents a feasible solution, and the processors have the entire knowledge of it. This is ensured by forbiding the moves that would render the knowledge incomplete or uncertain. Some syncronization phases and different partitioning schemes are used so as not allways forbid the same moves. In the chaotic strategy, the global state is still maintained coherent, but processors are allowed to perform moves having only an approximation of the global state. Therefore, in this case, computing the gain of a move is subject to an error. Infrequent syncronization phases where the global state is broadcasted to all the processors are used in order to not allow large errors.

The experiments and the benchmarkings we have presented on large sparse graphs show the viability and practibility of our approaches based on SS+SA. On one hand, using sequential SS+SA, we have found better approximations in radically less time. On the other hand, we have seen that the chaotic strategy for the parallel SS+SA has an excellent behaviour in terms of running time and solution quality compared to the sequential heuristic when running on a simple network of personal computers.

References

[1] J. Díaz, J. Petit, and M. Serna. A survey on graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

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

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

[4] J. Petit. Experiments for the MinLA problem. Report de recerca LSI-01-7-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, http://www.lsi.upc.es/~jpetit/Publications, 2001. Preliminar version in ALEX 98.

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

[6] B. Mohar and S. Poljak. Eigenvalues in combinatorial optimization. In R. A. Brualdi, S. Friedland, and V. Klee, editors, *Combinatorial and Graph-Theoretical Problems in Linear Algebra*, volume 50 of *IMA Volumes in Mathematics and its Applications*, pages 107–151, Berlin, 1993. Springer–Verlag.

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

[8] W. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

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

[10] R. Bar-Yehuda, G. Even, J. Feldman, and S. Naor. Computing an optimal orientation

of a balanced decomposition tree for linear arrangement problems. *Journal of Graph Algorithms and Applications*, 2002. To appear.

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

[12] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. *6th SIAM Conf. Parallel Proc. Sci. Comput.*, 1993.

Figures and tables

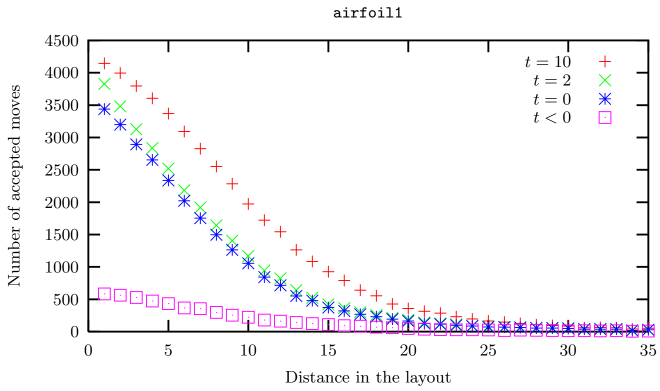


Figure 1: Number of accepted moves in function of their distance and the temperature (t) on the solution found by Spectral Sequencing for the `airfoil1` graph. ($t < 0$ means that only strictly descendent moves are accepted.)

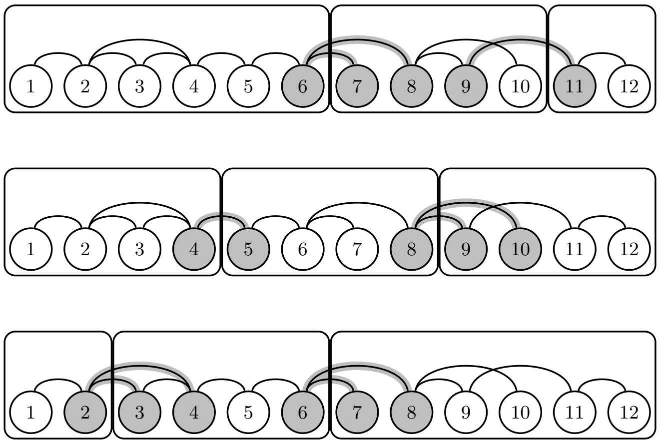


Figure 2: Examples of partitions for three processors: $\mathcal{V}_{+1}$ at the top, $\mathcal{V}_0$ at the middle and $\mathcal{V}_{-1}$ at the bottom.

Name	Nodes	Edges	Degree	Diam	Family
3elt	4720	13722	3/5.81/9	65	
airfoil1	4253	12289	3/5.78/10	65	FE
crack	10240	30380	3/5.93/9.00	121	
whitaker3	9800	28989	3/5.91/8	161	
c1y	828	1749	2/4.22/304	10	
c2y	980	2102	1/4.29/327	11	
c3y	1327	2844	1/4.29/364	13	VLSI
c4y	1366	2915	1/4.26/309	14	
c5y	1202	2557	1/4.25/323	13	
randomG4	1000	8173	5/16.34/31	23	Random geometric
mesh33x33	1089	2112	2/3.88/4	64	33×33-mesh

Table 1: Test suite. For each graph, its name, number of vertices, number of edges, degree information (minimum/average/maximum), diameter and family.

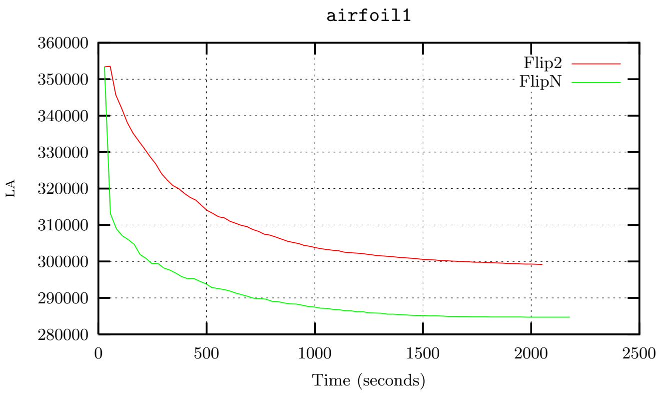


Figure 3: SS+SA on `airfoil1` using the Flip2 or FlipN neighborhoods.

	Cost			Time		
Graph	SS	SA	SS+SA	SS	SA	SS+SA
3elt	429164	428907	363686	2	9200	2564
airfoil1	353399	338120	285597	1	7427	2065
crack	1641119	1576500	1496671	4	46178	9310
whitaker3	1251709	1214090	1169642	3	41144	8127
c1y	104316	66894	63675	0	108	133
c2y	97218	84337	79429	0	183	174
c3y	181124	131022	123548	0	436	289
c4y	137701	123266	116140	1	514	293
c5y	144625	101498	103958	0	325	247
randomG4	197298	159370	151074	1	266	319
mesh33x33	36468	32605	31929	0	536	198

Table 2: Comparison between cost and time (in seconds) for Spectral Sequencing (SS), Simulated Annealing and sequential SS+SA.

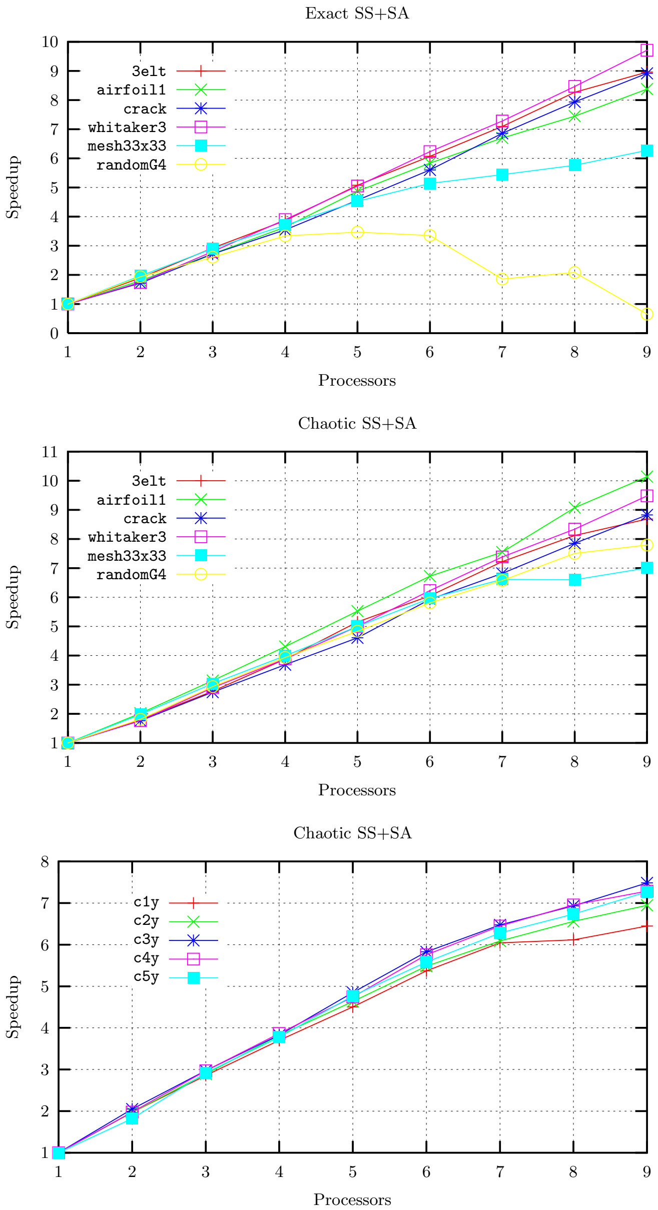


Figure 4: Speedup in function of the number of processors for the exact and chaotic SS+SA heuristics.

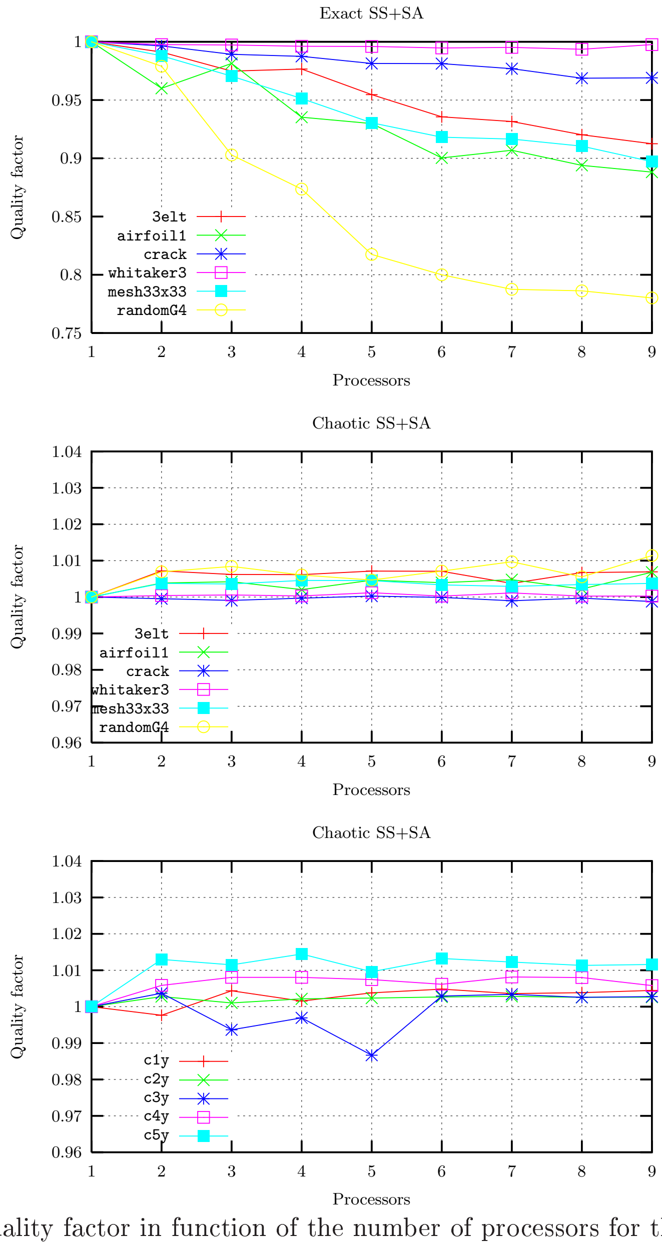
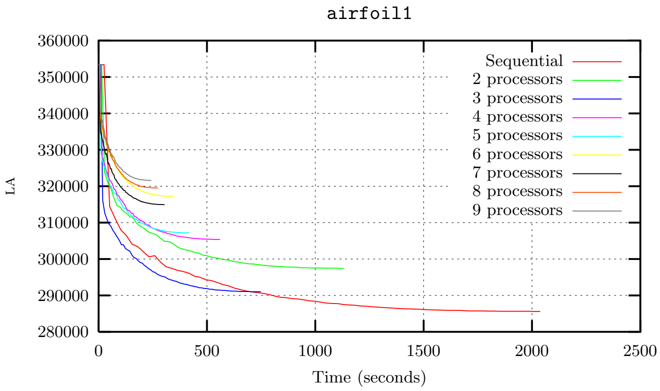
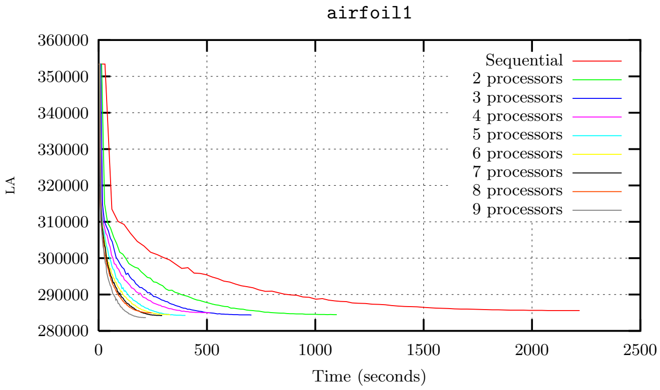


Figure 5: Quality factor in function of the number of processors for the exact and chaotic SS+SA heuristics.



(a) Exact parallel SS+SA



(b) Chaotic parallel SS+SA

Figure 6: Exact and chaotic parallel SS+SA on `airfoil1`: Cost in function of time depending on the number of processors.

Processors	1	2	3	4	5	6	7	8	9
`3elt`	1.000	0.991	0.975	0.977	0.955	0.936	0.932	0.920	0.913
	1.000	0.950	0.974	0.967	1.016	1.011	1.013	1.034	0.996
`airfoil1`	1.000	0.960	0.981	0.935	0.930	0.900	0.907	0.894	0.888
	1.000	0.901	0.908	0.910	0.974	0.973	0.957	0.931	0.931
`crack`	1.000	0.996	0.989	0.987	0.981	0.981	0.977	0.969	0.969
	1.000	0.859	0.907	0.887	0.913	0.934	0.980	0.992	0.991
`whitaker3`	1.000	0.998	0.997	0.996	0.996	0.995	0.995	0.994	0.998
	1.000	0.872	0.934	0.975	1.010	1.038	1.040	1.058	1.079
`mesh33x33`	1.000	0.988	0.971	0.951	0.930	0.918	0.916	0.910	0.897
	1.000	0.984	0.968	0.926	0.906	0.857	0.778	0.720	0.698
`randomG4`	1.000	0.979	0.903	0.874	0.817	0.800	0.788	0.786	0.780
	1.000	0.968	0.870	0.835	0.694	0.559	0.266	0.261	0.073

Table 3: Quality and efficiency factors for exact parallel SS+SA relative to sequential SS+SA. For each graph, the top number is the solution quality factor and the bottom number is the time efficency factor.

Processors	1	2	3	4	5	6	7	8	9
`3elt`	1.000	1.007	1.006	1.006	1.007	1.007	1.004	1.007	1.007
	1.000	0.900	0.926	0.969	1.031	1.009	1.032	1.015	0.965
`airfoil1`	1.000	1.004	1.004	1.002	1.005	1.004	1.005	1.002	1.007
	1.000	1.008	1.047	1.077	1.105	1.120	1.079	1.134	1.127
`crack`	1.000	1.000	0.999	1.000	1.000	1.000	0.999	1.000	0.999
	1.000	0.880	0.914	0.922	0.921	0.988	0.974	0.982	0.981
`whitaker3`	1.000	1.000	1.001	1.000	1.001	1.000	1.001	1.000	1.000
	1.000	0.885	0.967	0.970	1.000	1.038	1.055	1.042	1.054
`mesh33x33`	1.000	1.004	1.004	1.005	1.005	1.003	1.003	1.003	1.004
	1.000	0.991	1.013	0.998	1.000	0.994	0.945	0.825	0.779
`randomG4`	1.000	1.007	1.008	1.006	1.005	1.007	1.010	1.006	1.011
	1.000	0.906	0.978	0.984	0.967	0.968	0.940	0.938	0.866
`c1y`	1.000	0.998	1.004	1.002	1.004	1.005	1.004	1.004	1.004
	1.000	0.984	0.952	0.925	0.901	0.895	0.863	0.765	0.716
`c2y`	1.000	1.003	1.001	1.002	1.002	1.003	1.003	1.003	1.003
	1.000	0.984	0.964	0.955	0.927	0.914	0.870	0.820	0.772
`c3y`	1.000	1.004	0.994	0.997	0.987	1.003	1.003	1.003	1.003
	1.000	1.026	0.991	0.957	0.970	0.972	0.925	0.866	0.832
`c4y`	1.000	1.006	1.008	1.008	1.007	1.006	1.008	1.008	1.006
	1.000	0.988	0.989	0.967	0.949	0.959	0.921	0.869	0.810
`c5y`	1.000	1.013	1.012	1.014	1.010	1.013	1.012	1.011	1.012
	1.000	0.911	0.973	0.945	0.953	0.930	0.898	0.842	0.808

Table 4: Quality and efficiency factors for chaotic parallel SS+SA relative to sequential SS+SA. For each graph, the top number is the solution quality factor and the bottom number is the time efficency factor.