Benchmarking a Memetic Algorithm for Ordering Microarray Data

P. Moscato, A. Mendes, R. Berretta

Newcastle Bioinformatics Initiative
School of Electrical Engineering and Computer Science
Faculty of Engineering and Built Environment
The University of Newcastle
Callaghan, NSW, 2308, Australia

10 April 2006

Abstract

This work introduces a new algorithm for "gene ordering". Given a matrix of gene expression data values, the task is to find a permutation of the gene names list such that genes with similar expression patterns should be relatively close in the permutation. The algorithm is based on a combined approach that integrates a constructive heuristic with evolutionary and Tabu Search techniques in a single methodology. To evaluate the benefits of this method, we compared our results with the current outputs provided by several widely used algorithms in Functional Genomics. We also compared the results with our own hierarchical clustering method when used in isolation. We show that the use of images, corrupted with known levels of noise, helps to illustrate some aspects of the performance of the algorithms and provide a complementary benchmark for the analysis. The use of these images, with known high-quality solutions, facilitates in some cases the assessment of the methods and helps the software development, validation and reproducibility of results. We also propose two quantitative measures of performance for gene ordering. Using these measures, we make a comparison with probably the most used algorithm (due to Eisen and collaborators, PNAS 1998) using a microarray dataset available on the public domain (the complete yeast cell cycle dataset).

Key words:
Memetic Algorithms, Tabu Search, Gene Ordering, Clustering, Microarray

1 Introduction

Microarrays are undoubtedly one of the most exciting technological developments of the last decade (Gershon, 2002). The possibility to simultaneously measure the activity of the whole genome allows the creation of new methods to uncover large interacting biological systems. Using microarrays, scientists have now a tool to globally monitor the cell cycle and its regulatory mechanisms (Spellman et al., 1998). This raises hopes for a reliable molecular classification of cancers (Golub et al., 1999; Berretta et al., 2005) and for predicting the evolution of the disease from primary tumors (van't Veer et al., 2002; Moscato et al., 2005a,b). Microarrays may enable the prediction of cancer outcomes in which morphological classification is unreliable (Pomeroy et al., 2002). This small list of applications is illustrative of the promising aspects of this new technology which has certainly attracted a large number of

Email address:
{Pablo.Moscato,Alexandre.Mendes,Regina.Berretta}@newcastle.edu.au (P. Moscato, A. Mendes, R. Berretta).

scientists to investigate its strengths and weaknesses. A PubMed search with the word "microarray" shows that more than 14,000 articles appeared since its introduction.

In spite of its clear promises, some biologists are still doubtful about some of the findings obtained with this technology. The reason is the high variability of individual measurements which in turn may bias the data analysis. As a consequence, algorithm design for the analysis of microarray data should be made considering that it must be robust to the currently inevitable variability expected on its individual measurements.

In this paper, we analyze a problem that can generically be labelled as "gene ordering". More concretely, we are addressing the basic problem of visualizing in two-dimensions a large matrix of gene expression information. In some way, we are responding to the need of several biologists who are unsatisfied with the outputs of current commercial and public domain packages. These packages provide interesting and sometimes statistically sound clustering results (for instance k-means, hierarchical clustering and others (Eisen et al., 1998; Fasulo, 1999)), but they do not explicitly optimize their output in the way we do in this contribution. In some cases, closely similar gene expression patterns appear in the final display several thousand genes apart. This is also the case for entire clusters, as we will illustrate with both images and microarray datasets. In addition, one of the main objectives is to benchmark our results and to introduce some instances of the basic problem that would allow to study the robustness to the presence of errors in the measurements. This controlled experimentation allowed us to provide a metaheuristic method which is not sensitive to high levels of noise and highly varying individual measurements.

It is interesting to note that many clustering methods are basically some form of greedy algorithm, thus they can be trapped in suboptimal solutions of the objective function that they are implicitly or explicitly optimizing. Overcoming the limitations of these methods comes at a cost, and certainly it requires larger computer runs. A good example of an alternative method recently proposed was to use a Memetic Algorithm with a k-means agglomerative procedure as local search (Merz, 2003). In this work, we also present a Memetic Algorithm (MA) that uses a series of features that makes the search more effective. Our method can be seen as complementary to Merz's approach (Merz, 2003), since the main objective here is to provide an ordering of the genes, and not a clustering. However, we also introduce a novel hierarchical clustering algorithm to give an initial solution for our MA to speed-up the search. The main contributions in terms of memetic algorithm design are the use of a Tabu Search for local optimization and the definition of evolutionary operators which take advantage of the problem structure.

Four comparison algorithms are used to evaluate the performance of our MA.

The first one is a hierarchical clustering technique described in Section 3. In fact we use this procedure to initialize one of the solutions in our Memetic Algorithm. The second algorithm has been implemented in the software package called CLICK [1] (Sharan and Shamir, 2000). The authors reported better results in comparison to k-means and other traditional clustering techniques. Therefore it is worthwhile use CLICK for baseline comparison. The third algorithm is a hierarchical clustering created by the *European Bioinformatics Initiative* (EBI) as part of the *Expression Profiler* software tool [2]. Finally, the fourth method was proposed by Eisen et al. (1998) [3], which is a hierarchical clustering algorithm that also performs the ordering of the genes. These methods were chosen because of their wide acceptance by other researchers (Bini et al., 2003; Brazma and Vilo, 2000; Elkon et al., 2003; Wingender et al., 2000; Shamir and Sharan, 2002; Raffelsberger et al., 2002; Vilo et al., 2003). The second, third and fourth methods are currently available in the internet for download or online use.

In addition to the new method proposed, we address the problem of performance comparison. This is a critical point, since there is no unique measure for the solution quality in the ordering problem (Eisen et al., 1998; Tamayo et al., 1999). In order to evaluate the algorithm, we propose to complement existing comparison approaches with the use of images. Such images are corrupted by noise and the rows and columns are subsequently permuted at random, allowing a controlled benchmark to test the robustness of the algorithms. Their characteristics are explained later in Section 5. The algorithms were also tested on two microarray instances of interest – Fibroblast (Iyer et al., 1999) and Yeast (*Saccharomyces cerevisiae*) (Eisen et al., 1998) – with 517 and 6,221 genes, respectively. Our results show that some algorithms seem to scatter groups of genes that could have been placed relatively closer in the final image. This, in turn, may indicate that some functional groups may have a larger number of genes than previously reported, as our results on Yeast may indicate. This means that entire clusters with similar activity patterns can be closer in the final layout.

The paper is organized as follows. In Section 2, we describe the gene ordering problem. Section 3 presents the hierarchical clustering technique implemented to seed our Memetic Algorithm, described in Section 4. The instances are described in Section 5 and finally, computational results and conclusions are presented in Sections 6 and 7, respectively.

[1] `http://www.cs.tau.ac.il/~rshamir/expander/expander.html`
[2] `http://ep.ebi.ac.uk/EP/EPCLUST/`
[3] `http://rana.lbl.gov/EisenSoftware.htm`

2 The gene ordering problem

The input of the gene ordering problem under consideration is defined as an integer matrix G of gene expression values g_{ij} where $1 \leq i \leq n$ and $1 \leq j \leq m$, such that n is the number of genes, m is the number of experiments/conditions, and g_{ij} represents the expression level of gene i under condition j. Informally, the task is to find a permutation of the genes' names $\pi = (\pi_1, \pi_2, ..., \pi_n)$, such that genes with similar expression are positioned close to each other in the sequence.

In order to measure the degree of similarity between two genes we can use the Euclidean distance or other measures of similarity as those based on a Pearson correlation. A key point to evaluate the quality of an ordering sequence is to choose a suitable objective function. Our objective function is a sum of n partial sums, one for each gene (Cotta et al., 2003). Each partial sum considers the gene as the center of a "window" and a partial cost calculation considering the closest genes inside that window is done. The partial cost for a gene l is calculated as:

$$\sum_{i=\max(l-w_s,1)}^{\min(l+w_s,n)} (w_s - |l-i| + 1).D[\pi_l, \pi_i] \tag{1}$$

where the window size is $2w_s + 1$ (the number of genes involved in each partial distance calculation) and $D[\pi_l, \pi_i]$ represents the distance between genes π_l and π_i. A weight $(w_s - |l-i| + 1)$ was added to Eq. 1 to give more importance to the genes that are closer. For example, consider the gene in the position $l = 5$ and $w_s = 2$. Then, the partial sum for this gene will be: $D[\pi_5, \pi_3] + 2D[\pi_5, \pi_4] + 2D[\pi_5, \pi_6] + D[\pi_5, \pi_7]$. Note that the window is 5, so, for π_5 we are considering in the sum π_3, π_4, π_6 and π_7, but with different weights. Therefore, the $TotalCost$ is calculated as

$$TotalCost(\pi) = \sum_{l=1}^{n} \sum_{i=\max(l-w_s,1)}^{\min(l+w_s,n)} (w_s - |l-i| + 1).D[\pi_l, \pi_i] \tag{2}$$

Concerning the parameter w_s, previous tests in Cotta et al. (2003) indicate that the best results seemed to be obtained when $w_s = 5\%$ of the number of genes, and we have also used that value in this contribution. However, in Section 7, we provide a quantitative analysis of the influence of this parameter. We also show the characteristics of solutions returned by our MA when this parameter is modified. We have found that this parameter can be user-defined and that different choices will give different solutions to the end user.

3 The agglomerative hierarchical clustering algorithm

In general, methods for gene ordering use some type of bottom-up agglomerative constructive heuristic. We decided do design an algorithm of that type as a starting point for the MA. Therefore, the MA will try to improve the solution obtained by the hierarchical clustering technique after adding its solution to the initial population.

Hierarchical clustering techniques have been applied successfully for gene clustering in recent years (Eisen et al., 1998; Biedl et al., 2001; Martin et al., 2001). They are generally very fast and provide reasonable solutions of interest for a preliminary visualization of the results. However, as greedy procedures, their performances are not competitive against more sophisticated metaheuristics.

Method hierarchicalClustering
begin
  $cluster = \{1, 2, ..., n\}$
  while($|cluster| > 1$)
    selectMostSimilar($clusterA$,$clusterB$) $\in$ $cluster$;
    $clusterA$ = **joinClusters**($clusterA$,$clusterB$);
    removeCluster($clusterB$);
  end
end

Fig. 1. Pseudo-code for the hierarchical clustering technique.

Our hierarchical clustering technique provides the gene ordering and an associated hierarchical tree (Figure 1). The algorithm begins with n clusters, each one containing one gene only. The algorithm then chooses the two clusters that have the minimum distance between each other. As we are interested in obtaining a sequence and an associated hierarchical tree, we examine the four

Step 1

	1	2	3	4
1	---	2.4	1.7	6.8
2	2.4	---	3.6	6.4
3	1.7	3.6	---	2.0
4	6.8	6.4	2.0	---

Step 2

	1'	2	4
1'	---	3.0	4.4
2	3.0	---	6.4
4	4.4	6.4	---

Step 3

	1''	4
1''	---	5.1
4	5.1	---

Step 1: Join 1 and 3
Cluster 1 becomes 1' = 1+3
Recalculate distances: d(1',2) = [d(1,2)+d(2,3)] / 2
d(1',4) = [d(1,4)+d(3,4)] / 2

Step 2: Join 1' and 2
Cluster 1' becomes 1'' = 1'+2
Recalculate distances: d(1'',4) = [d(1',4).2+d(2,4).1] / 3

Step 3: Complete the binary tree by joining 1'' and 4.

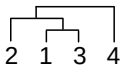

Fig. 2. Example of the hierarchical clustering with four genes.

possible ways of joining the two clusters by their extreme points. The algorithm joins the clusters by the extremes where the genes have the minimum distance between them. The distances between the new cluster and the others will be the average of the original ones, weighted by the number of genes in each one of them (see example in Figure 2). The method stops when there is only one cluster, containing all genes. The pseudo-code in Figure 1 describes the technique. In Figure 2, we show an example with four genes to illustrate how the algorithm constructs a solution.

4 The memetic algorithm

Memetic Algorithms (Moscato, 1989, 1993; Moscato and Norman, 1992) are population-based search methods for optimization. They form a type of metaheuristic that balances *exploration* and *exploitation* well to find high-quality solutions of the optimization problem at hand. Exploration is the capacity of a randomized algorithm to sample different areas of the search space, looking for the most promising regions. Once such regions are detected/selected or randomly sampled, exploitation takes place, trying to find high-quality solutions. Exploration is obtained by means of the population approach together with the use of recombination and mutation operators. Exploitation comes with the use of individual improvement procedures, usually efficient local search methods (Buriol et al., 2004). In our MA, we have decided to use the Tabu Search metaheuristic for exploitation. Other forms of individual optimization methods, like truncated exact algorithms, are also used in MAs (Puchinger et al., 2005; Klau et al., 2004; Cotta and Troya, 2003).

Figure 3 shows the pseudo-code of the memetic algorithm that we have implemented. The algorithm begins by initializing the population. In the first generation, all solutions are generated at random with the exception of one, the solution obtained by the agglomerative hierarchical clustering technique (described in Section 3). The evolutionary loop follows, new solutions are created, modified and inserted into the population. After a given number of generations, we apply different procedures that improve the solutions. Finally, we restart the population and go back to the evolutionary loop again. We will now explain each one of the memetic algorithm's components.

4.1 Population Structure

The use of structured populations, in particular hierarchically structured, improves the performance of evolutionary/memetic algorithms (Moscato and Norman, 1992; Buriol et al., 2004; Franca et al., 2001; Berretta and Moscato,

```
Method memeticAlgorithm
begin
  initializeSolPopulation(pop);
  updatePopStructure(pop);
  repeat
    for i = 1 to numberOfRecombinations do
      selectSolutions(solutionA, solutionB) ⊆ pop;
      newSolution = recombine(solutionA, solutionB);
      newSolution = mutate(newSolution);
      insertSolution(newSolution, pop);
    end
    updatePopStructure(pop);
    if (populationHasConverged pop) then
      improveSolution(pop);
      checkForNewIncumbent(pop);
      updatePopStructure(pop);
      initializeSolPopulation(pop);
    end
  until (stopCriterion);
end
```

Fig. 3. Pseudo-code for the memetic algorithm.

1999) and is fundamental for the recombination phase, where pairs of solutions are selected using information about the positions they occupy in the population structure.

The population is structured as a ternary tree with 13 agents (Figure 4) which can be understood as a set of four overlapping sub-populations. Each subpopulation is composed of one *leader* and three *supporters*, which are one level below in the hierarchy. Agent 1, the root of the tree, is the leader of the top subpopulation and it has agents 2, 3, and 4 as supporters. We note that agent 2 has as supporters agents 5, 6, and 7, and so on. So the agents in the second level act as leaders and supporters in different subpopulations.

Each agent in the population has two solutions, namely *pocket* and *current*.

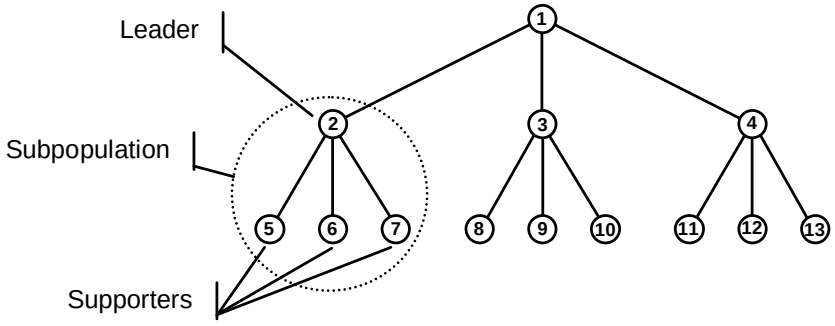

Fig. 4. Diagram of the population structure.

Whenever within an agent the *current* solution is better than the *pocket*, they are switched. In addition, the *leader* agent is always better than their *supporters*. The procedure **updatePopStructure** in the pseudo-code enforces that. These two mechanisms guarantee the flow of the best solutions towards the agent at the top of the hierarchy.

4.2 Representation

Solutions of the gene ordering problem are represented as in our hierarchical clustering – a binary tree whose leaves are the genes. Figure 5 shows an example of a solution with six genes. We have used a preorder traversal of the associated binary tree to represent the solution. The idea of the preorder traversal is to have the information of the root followed by the left and the right subtrees respectively. We used a string of integers to code solutions, where each gene is identified with a number in the interval [1, n] and internal nodes of the tree with the value -1 (all internal nodes are indistinguishable). For instance, in Figure 5, the first '-1' represents the root, the next '-1' represents the first node on left and finally the '4' represents the first leaf.

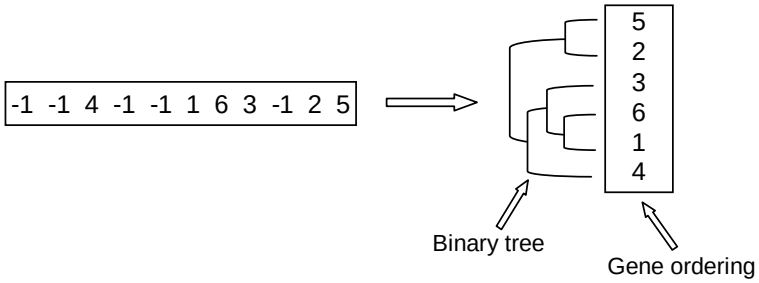

Fig. 5. Pre-order traversal representation of a solution.

4.3 Recombination and mutation

Recombination and mutation must be cleverly designed as they have very important roles in the dynamics of the MA algorithm. Recombination creates new solutions from solutions already visited by combining characteristics of them. On the other hand, mutation helps to explore from an individual solution, allowing that new characteristics can be incorporated in the population.

The *recombination* algorithm receives as input two solutions based on their position in the ternary tree. It uses the *pocket* solution of a *leader* and one *current* solution randomly selected from its *supporters*, following similar guidelines of previous implementations that have the same approach (Buriol et al., 2004; Franca et al., 2001; Berretta and Rodrigues, 2004). It then selects

a subtree from the first solution, lists all the genes inside it and removes them from the second solution, as no duplicated genes are allowed. Next, we have the completion phase, where the selected subtree is inserted into a random position of the second solution. The subtree can be inserted both in the right or in the left ramification. In Figure 6, these two possibilities are described.

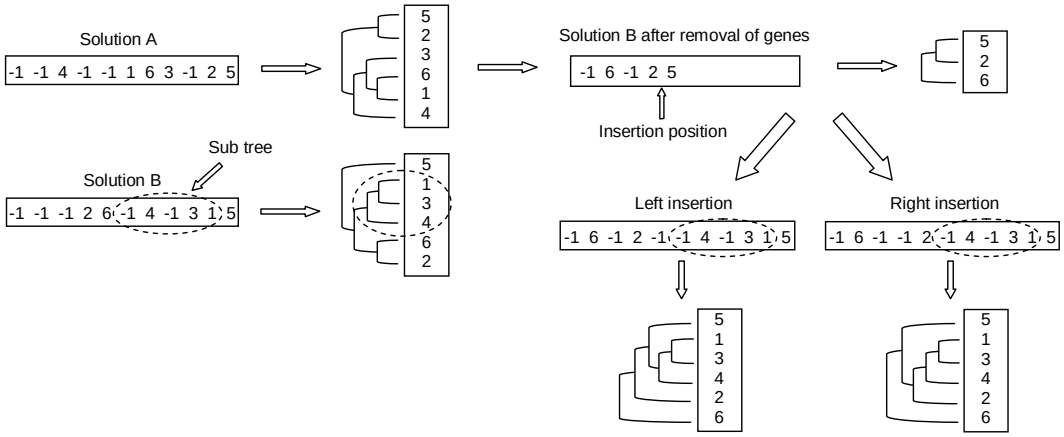

Fig. 6. Example of a subtree-based recombination. A subtree of *solution B* is selected and inserted into *solution A*, in the left or in the right part of the ramification. All choices – subtree selection, insertion position and left/right insertion – are at random.

The **mutation** is based on a randomized sequence flip. A subtree is selected uniformly at random and the entire sequence of genes belonging to it is flipped. This mutation preserves the gene grouping inside the subtree and thus creates a relatively minor perturbation. The tree structure is also flipped, to keep consistency with the sequence reversal. After recombination and mutation, the new solution will replace the *current* solution in the *supporter* agent that participated in the recombination.

4.4 Improvement Procedures

We implemented three different procedures to improve the solutions. They are all based on local search, but in two of them we use Tabu Search. We will start by describing the first two Tabu Search improvement procedures.

Tabu Search (TS) (Glover and Laguna, 1997) is a metaheuristic with several successful implementations tackling hard combinatorial problems. Its key strategy is to guide a local search procedure using adaptive memory to prevent reaching solutions previously visited and consequently, to explore new solutions avoiding local optimality (Glover and Laguna, 1997). The use of TS inside the local search and as a diversification mechanism in Memetic Algorithms has been tried before with very excellent results in several problem domains (Moscato, 1993; Berretta and Moscato, 1999).

A Tabu-based local search starts with a single solution s. This initial solution is iteratively replaced by another, say s', obtained from a set $N(s)$, which represents the solutions that are neighbors of s. $N(s)$ is composed of all solutions obtainable from s by some perturbation or movement. TS changes the neighborhood size by classifying certain moves as forbidden or *tabu*. This mechanism is called *adaptive memory* and keeps track of *solution attributes* that have changed during the recent past. The use of memory prevents cycling, i.e., the indefinite execution of the same sequence of moves, and directs the search to unexplored regions. It is also possible to add an *aspiration criterion* which removes the tabu condition of a move if it is considered attractive at that moment of the search.

In our first procedure, **tabuSearchSwap** (Figure 7), a move is defined by swapping the positions of the genes. First, the procedure chooses at random a position i. Next, the method sequentially selects a position j, belonging to the interval $[i-w, i+w]$, where w is a parameter to be chosen. This interval aims to reduce the neighborhood, since it becomes prohibitively time-consuming to test all possible swaps when the number of the genes is large. A swap move is confirmed if it improves the solution and both i and j are not *tabu*. If such move does not exist for any j, we perform the move that least worsens the solution. Anytime a move occurs, i and j become *tabu* for a number of iterations. Figure 7 shows the pseudo-code for the **tabuSearchSwap** procedure, where $f(s)$ represents the objective function of a solution s.

```
Method tabuSearchSwap(initialSolution)
begin
  s = initialSolution;
  repeat
    select(i) ∈ {1, ..., n};
    J = {i − w, ..., i + w};
    while(J ≠ ∅ or noMovementAccepted)
      select(j) ∈ J; J=J - {j};
      s' = swap(i,j);
      if ((f(s') < f(s) and isNotTabu(i,j)) or aspirationCriteria(i,j))
        s = s'; makeTabu(i,j);
        updateIncumbent();
      end
    end
    if (noMovementAccepted)
      select(j') ∈ {i − w, ..., i + w};
      s = swap(i,j'); makeTabu(i,j');
    end
  until (stopCriterion);
end
```

Fig. 7. Pseudo-code of the swap-based tabu search.

```
Method tabuSearchTree(initialSolution)
begin
  s = initialSolution;
  repeat
    numTries = 0; W = ∅;
    while (numTries < maxTriesWithoutImprovement)
      numTries++;
      select(i) ∈ Nodes (set of nodes from the tree);
      s' = flipTree(i);
      if ((f(s') < f(s) and isNotTabu(i)) or aspirationCriteria(i))
        s = s'; numTries = 0; makeTabu(i);
        updateIncumbent();
      else W = W + {i};
    end
    select(i) ∈ W;
    s' = flipTree(i); makeTabu(i);
  until (stopCriterion);
end
```

Fig. 8. Pseudo-code of the tree-flip-based tabu search.

In the second procedure, called **tabuSearchTree** (Figure 8), a movement is defined by flipping a subtree. The procedure starts choosing at random a node from the tree. If flipping the elements of this tree improves the solution the flip is performed and the node becomes *tabu*. After some – namely 5% the number of genes – attempts without success, the procedure selects the least worse move from a set W and the node becomes *tabu*. This set starts empty and each time a flip is tested and it is not done, the move is stored in W.

In both procedures (**tabuSearchSwap** and **tabuSearchTree**) the *aspiration criteria* is the following: a tabu move is considered not tabu if the solution reached by this move is better than the incumbent. The tabu tenure – the number of moves the variable remains *tabu* – was determined after preliminary trial-and-error runs on different types of instances of several sizes, and we finally adopted a value proportional to the number of genes. In the implementation, the tabu tenure is a random value lower than 5% of the number of genes. Since the TS acts as a complementary technique to the MA, we have restricted the number of iterations in which we run the TS, thus balancing the fraction of CPU time dedicated to it. The number of iterations was set at ten times the number of genes. This value allows the TS to reach a local minimum, select a series of tabu movements and reach other local minima.

The third local search, called **moveBlocks**, aims at approximating groups of genes (*blocks*) which may be far apart in the current sequence order. Intuitively, a block begins (or ends) every time two consecutive genes have very different expression profiles (see Figure 9). The algorithm searches the k most different pairs of adjacent genes in the solution and consider them as block

frontiers. As there is no direct way to predict how many groups of genes could be useful to move, we vary this parameter k between 3 and 50, increasing/decreasing it by one, every time the local search procedure is called. Figure 9 illustrates a typical *Lenna* solution in a case where such groups can be easily spotted.

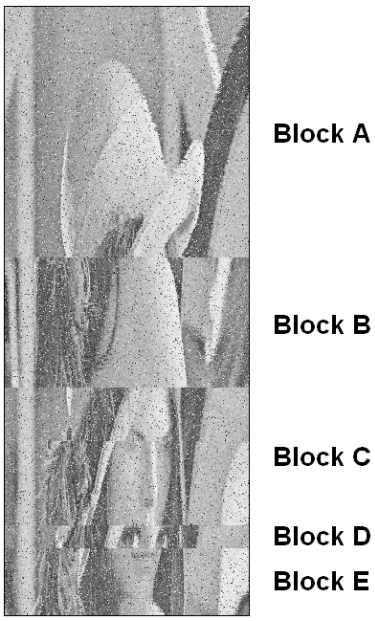

Fig. 9. A local search algorithm is used to move "blocks" which are identified as consecutive pairs of highly dissimilar expression patterns. The figure illustrates a situation in which five blocks are clear and an *ad hoc* local search method is highly beneficial.

After the local search identifies the blocks' frontiers, it iteratively performs a series of trial movements – inversions, insertions and swaps – always aiming to find a solution with less discontinuities. A pseudo-code is shown in Figure 10.

```
Method moveBlocks(initialSolution)
begin
  s = initialSolution;
  repeat
    findBlocks(s,k);
    s = invertBlocks(s);
    s = insertBlocks(s);
    s = swapBlocks(s);
  until (noImprovement);
end
```

Fig. 10. Pseudo-code of the block-based local search.

5 The sets of instances

A common difficulty of gene ordering and clustering algorithms applied to microarrays is how to compare two solutions. In general, they are evaluated by visual inspection which, in our experience, is highly subject to misjudgments.

We can make an analogy between ordering genes in microarray data and the problem of "unscrambling" the rows of an image when the image has all its rows permuted at random. This means that, it would be possible to use images as benchmark instances to evaluate gene ordering and clustering algorithms. The advantage of using images for performance evaluation is that it becomes easier to understand the quality of the results. With these ideas in mind, we introduced five sets of instances. The first three use images and the other two use microarray data. All sets are described next.

5.1 'Lenna'*-based instances*

Since the beginning of the 70's, many computer-vision articles adopted this image for evaluation purposes of algorithms and it is now considered one of the most famous pictures in Image Processing [4]. The original Lenna image has 512x512 pixels, but since most microarrays experiments involve a larger number of genes, we replicated it 10 times, creating a 'multiplied' Lenna image with 5,120x512 pixels. In the original image each row is very similar to its neighbors and there are no repeated row patterns. Thus, an objective function that aims to group genes with similar gene expression profiles should consider the original order of the lines as optimal or very near to the desired optimal order. When we scramble the rows of the 'multiplied' image and then try to order them again, we expect to get back the original one, but as the image was replicated 10 times, the optimal ordering would resemble a 'stretched' Lenna – or a "*Modigliani* Lenna" (recalling the Italian artist's elongated portraits).

To simulate the level of error of microarray experiments (Kothapalli et al., 2002), we added noise to the image in two different ways. The pictures are gray-scaled, and their pixels' intensity varies between 0 and 255 (see Figure 11). The first type of noise, namely *Type I*, modifies each image pixel by a random value lower than x% of the maximum pixel intensity - i.e. x% of 255. The second noise, namely *Type II*, assigns a random value to x% of the picture's pixels. Both types of error were applied together at 10%, 20%, 30% and 40% to create the first four test images.

[4] `http://www.public.asu.edu/~akandan/tech/lena/`
`http://www.cs.cmu.edu/~chuck/lennapg/lenna.shtml`

Most microarray datasets contain no more than a few dozen experiments, due to their high costs, and we are aware that with the current costs of this technology the use of an instance with 512 columns would reflect a rather unrealistic situation. However datasets of this sample size already exist and may become common place in the near future (Gunderson et al., 2004). In order to discuss current, more typical instances of this problem, in the next four instances we selected 100 adjacent columns from the original image, centered in the region of the eyes, a band that holds the majority of the conspicuous characteristics of the image. As expected, this second set of instances became more challenging for all the algorithms. Overall, the results got worse for the

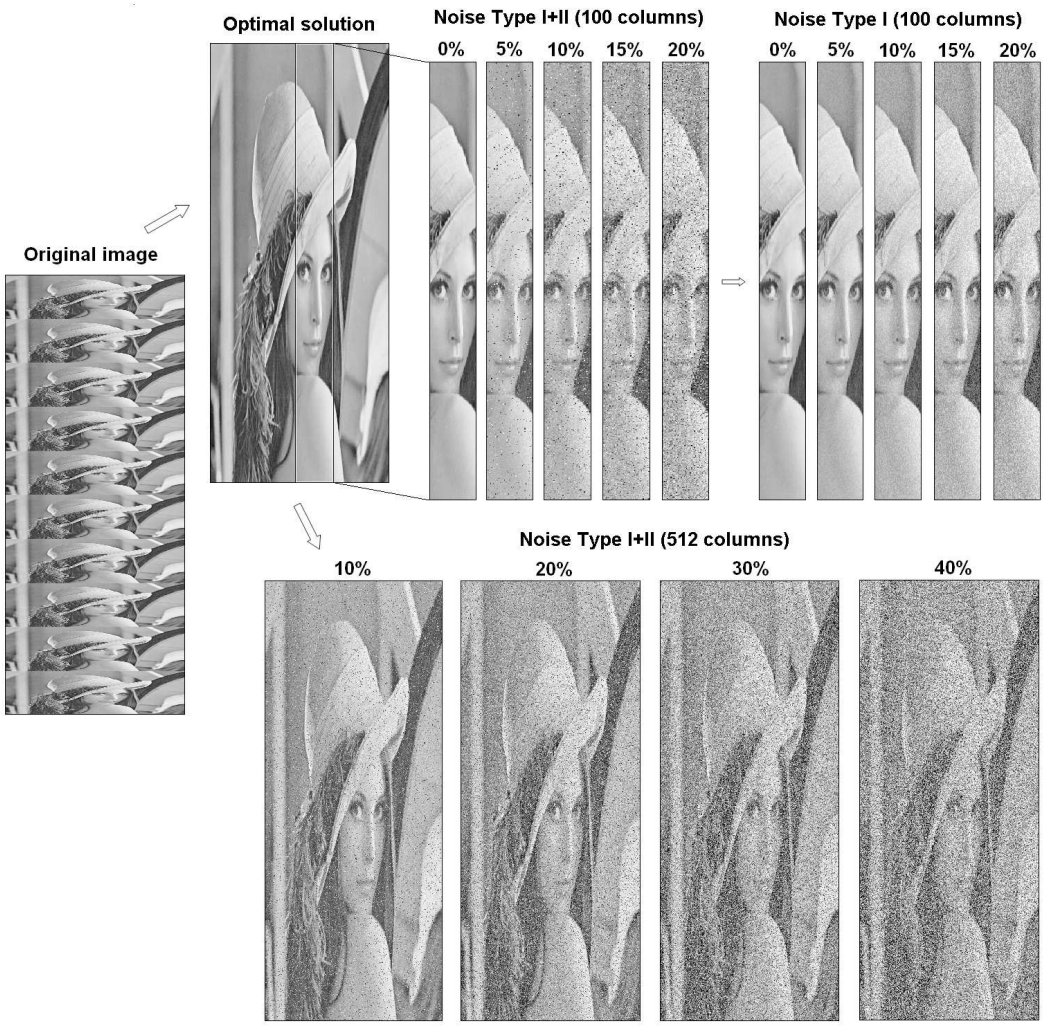

Fig. 11. Instances derived from the Lenna 512x512-pixel image. On the left, the *multiplied Lenna* which is ten copies of the original image. On the top are the expected optimal solution, *striped Lenna* with *Type I+II* noise and only noise *Type I*, respectively. On the bottom, four *Whole Lenna* with *Type I+II* noise. This allows a controlled benchmark, as all algorithms will then receive a dataset on which all the rows have been permuted at random.

same values of *Type I+II* noise levels. This indicates that it is relevant to study the performance of the methods in detail for lower noise levels (5%, 10%, 15% and 20%).

The third set of instances is composed of the same 100 adjacent columns of the second set, but contains only noise *Type I*. The motivation is that the impact of *Type II* noise in the performance of the algorithms can be minimized by a posterior re-analysis of any suspicious measurement by a repeated experiment. In Figure 11 we present some instances derived from the 'multiplied' Lenna image.

5.2 The Fibroblast *and* Yeast *instances*

The biological instances were used to check the performance of the algorithms on real microarray data with similar numbers of genes. The first instance (*fibroblast*) has 517 genes corresponding to the human fibroblast response to serum (Iyer et al., 1999). This instance was chosen because it is on the same range of the original Lenna image, the red/green clusters in the resulting images are easier to perceive, we have worked with it in the past and it seems to be a suitable data set for visual comparison.

The second biological instance derives from gene expression studies of the yeast cell-cycle (*Saccharomyces cerevisiae*) with 6,221 genes, firstly introduced by Eisen et al. (1998) and still considered a standard benchmark. This dataset is available for download from numerous websites [5], and was also used previously (Wu et al., 2002; Teichmann and Babu, 2002).

6 Comparison methods

We compared our results with three other methods. The first one is the *CLICK* algorithm (Sharan and Shamir, 2000). It was referenced many times since its publication and it is widely recognized as a high-performance method (Elkon et al., 2003; Wingender et al., 2000; Shamir and Sharan, 2002). We will briefly describe it. *CLICK* initially calculates a *similarity matrix* between genes. From this similarity matrix, *CLICK* computes a weighted similarity graph $G = (V, E, w)$, where w reflects the probability of two genes to be in the same cluster. Then, it recursively solves a series of MIN CUT problems producing a set of subgraphs, each one representing a cluster. At the end, two steps, namely

[5] Please refer to the web sites:
`http://www.yeastgenome.org/`
`http://cellcycle-www.stanford.edu/`

adoption and *merging* steps, decide merging clusters until a given threshold is reached.

The second algorithm is a hierarchical clustering from the *European Bioinformatics Initiative* (EBI) software named *Expression Profiler*. We refer the reader to a recently published book for more details about it (Vilo et al., 2003). The software is also widely referenced in the literature (Bini et al., 2003; Brazma and Vilo, 2000; Raffelsberger et al., 2002). The Expression Profiler allows you to choose from many types of algorithms, as well as from different distance metrics. For the tests, we chose the hierarchical clustering with an *average weighted group linkage* option and *Euclidean distances squared*, which seemed the combination that helps to obtain the best results.

The third algorithm was introduced by Eisen et al. (1998). This is one of the most used algorithm in Bioinformatics and the original publication has been already cited more than 2,800 times. For the comparison, we used the solutions with the best hierarchical clustering, obtained using *centered correlation* and *average linkage* clustering. The software is available for download from the EisenLab Homepage [6] – Gene Cluster V2.11.

7 Computational results

In this section we present the results obtained by the algorithms on the five sets of instances described in Section 5. All tests (except the hierarchical clustering from EBI, which was done online using their server) were performed on a Pentium IV 3.0 GHz HT computer, with 1 Gb of RAM, and using Java 1.4.2 JDK.

In Sections 7.1, 7.2 and 7.3, we compare our methods against CLICK and EBI hierarchical clustering. Then, in Section 7.4, we extend the comparison to Eisen's hierarchical clustering, where we also use a quantitative measure for the quality of the solutions found. We highlight some interesting results on the identification of some functional groups in Yeast. CPU times are reported in Table 1.

7.1 Results for the Whole Lenna *image*

In Figure 12 we show the solutions found for the *Whole Lenna* images with noises *Type I+II* at levels ranging from 0% to 40%. A few considerations must be taken into account. The CLICK algorithm focuses only in finding

[6] http://rana.lbl.gov/EisenSoftware.htm

Table 1
CPU times for the methods

Instance	Number of genes/exp.	Agglom. Clustering	Memetic Tabu	Agglom. Clustering EBI	CLICK
Whole Lenna	5,120/512	25 sec	120 min	60 sec	3 min
Striped Lenna	5,120/100	25 sec	120 min	60 sec	3 min
Fibroblast	517/18	1 sec	3 min	1 sec	25 sec
Yeast	6,221/80	38 sec	150 min	80 sec	4 min

good clusters, and lacks a within-cluster ordering procedure which may be an interesting add-on for that approach. Such lack of ordering is troublesome when it comes to analyze individual genes (as similar genes could be scattered between clusters). As the largest clusters obtained by CLICK contained several hundred genes, such analysis is complicated, even considering that they have similar expression profiles. An embedded hierarchical clustering could create a better within-cluster ordering and CLICK's solution would be easier to analyze.

The EBI hierarchical clustering had a good performance. It found most of the characteristics, but completely missed the overall distribution of patterns. Its performance seems to be independent of the noise level. A strong point is that the algorithm is very fast and thus can be used as a starting point for a more powerful metaheuristic, like the one being proposed in this paper. Our agglomerative hierarchical clustering also obtained good results with up to 30% noise and was the fastest method. However, we must emphasize that since the EBI algorithm runs directly on a dedicated web site, we do not have information about processor speeds, whether any parallel processing technique is being employed, or any other details that might influence the observed CPU time. Finally, the MA algorithm returned the best results of them all, restoring near-optimal solutions with up to 40% noise, at the expense of a longer CPU time.

7.2 Results for the Striped Lenna *image*

In Figure 13 we show the solutions for the *Striped Lenna* images with noise *Type I+II* at levels up to 20%. These images were more difficult to order than the previous ones.

The *Striped Lenna* results exhibited the same pattern of the *Whole Lenna*. The CLICK algorithm obtained good clusters, but without any intra-cluster optimization, it is difficult to accurately evaluate the results. Although the two

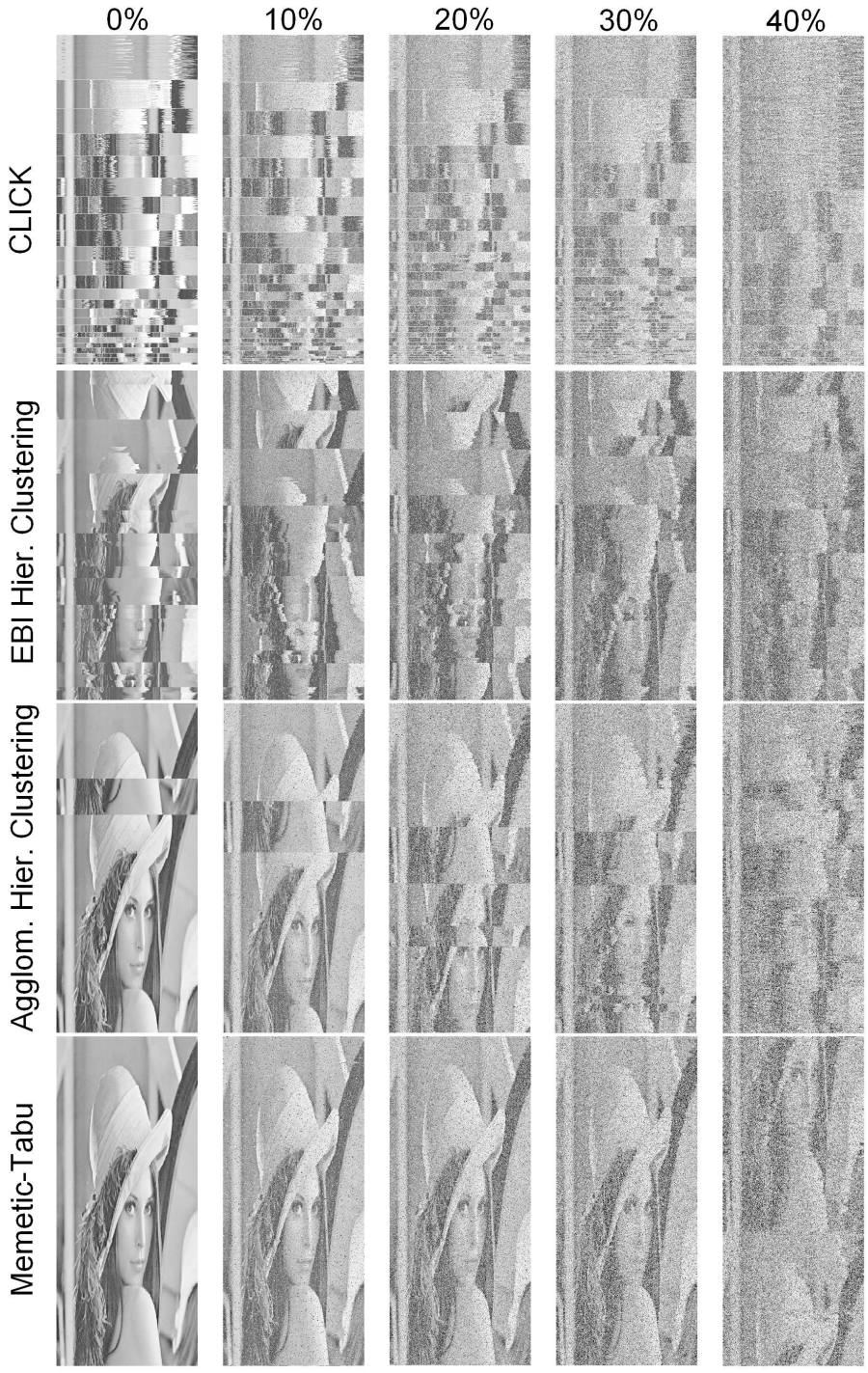

Fig. 12. *Whole Lenna* instances results for the four algorithms. Noise level varies from 0% to 40%.

hierarchical clustering algorithms' performances got worse, the EBI algorithm obtained slightly better solutions. The MA recovered the image with no noise, and with 5% noise the results were also very impressive. For higher noise levels the performance of MA rapidly deteriorated, but it still obtained the best results.

In Figure 14 we show the solutions found for the *Striped Lenna* images with only noise *Type I* from 0% to 20%. The images are easier to order compared to the ones in Figure 13. This test also illustrates how noise *Type II* is potentially very harmful for some algorithms.

7.3 Results for the Fibroblast *and* Yeast *instances*

The *fibroblast* and the *yeast* instances were used to test the algorithms with real microarray datasets. Due to the expression profiles and smaller number of genes in the *fibroblast* instance, it is easier to make conclusions based on visual assessment only. On the other hand, with the *yeast* dataset it is very hard to tell which method is the best. We will give a quantitative analysis for the latter. In Figure 15, we show the orderings we obtained.

Although the images in Figure 15 are very similar, at first glance the CLICK appears to be the best method because it seems to have better defined groups. We will show that this is not the case. If we take the MA and the CLICK solutions for the *fibroblast* instance (see Figure 16), the upper half of both clusterings roughly contains the same genes in different order. Next to the MA solution (*center-left*), we highlighted its upper 280 genes, in the same order. If we randomize the positions of these 280 genes we get a new configuration that visually resembles CLICK's ordering (*center-right*). What this means is that,

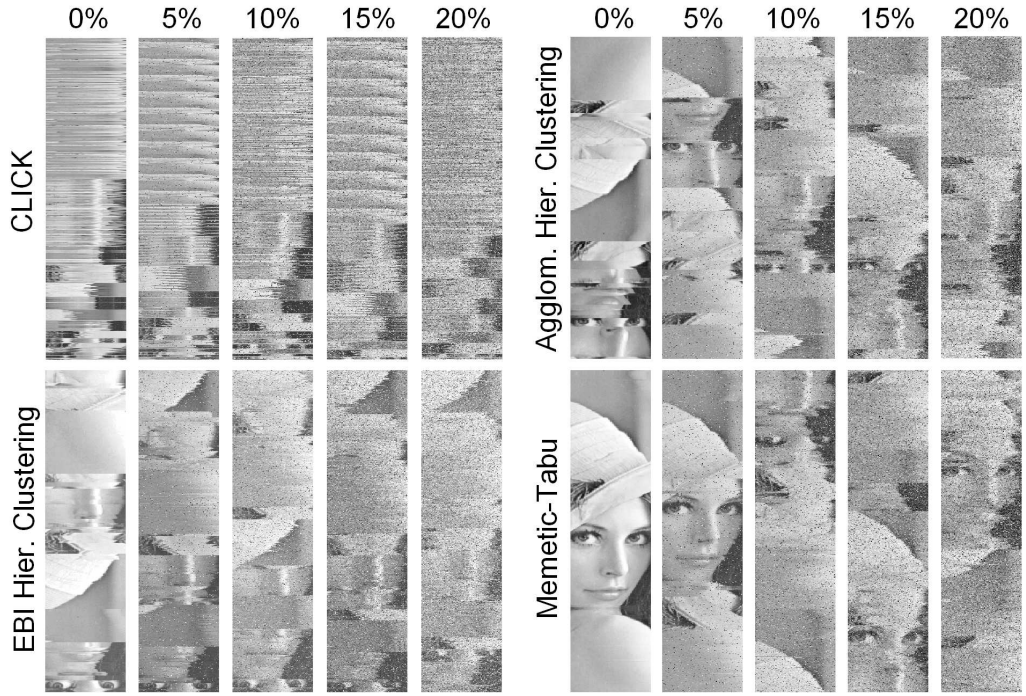

Fig. 13. *Striped Lenna* instances results for the four algorithms. Noise *Type I+II* level varies from 0% to 20%.

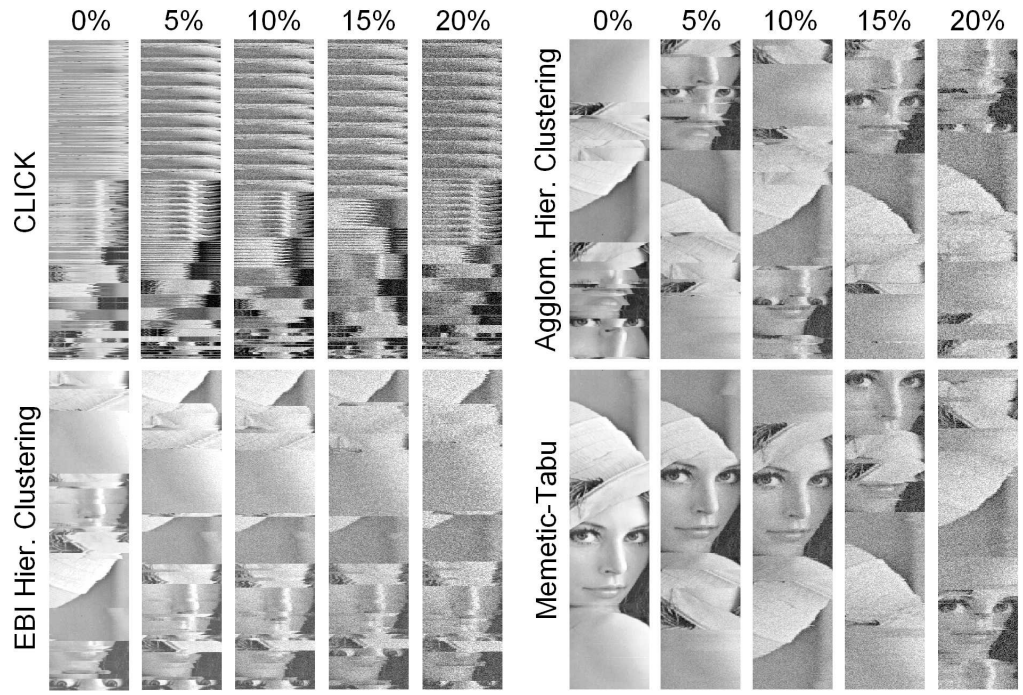

Fig. 14. *Striped Lenna* instances results for the four algorithms. Noise *Type I* level varies from 0% to 20%.

sometimes, a completely random ordering seems, visually, correlated enough, and visual judgements alone could be potentially misleading. The good visual results obtained by CLICK – with blocks apparently well defined – might result in part from the lack of order within each cluster. This is an example of how visual evaluation for microarray orderings is a common trend that provides a weak criterion. It also shows the need of quantitative measures for comparison of algorithms and the use of controlled experiments, such as those we propose here, to evaluate the performance of algorithms.

7.4 Comparison between the memetic algorithm and Eisen's hierarchical clustering

In this section, we complement the previously proposed visual evaluation of different ordering methods by using a quantifier based on the correlation between genes. Additionally, we use the concept of *cliques* – from Graph Theory – as an indicator of groups of highly-correlated genes. Considering an undirected graph $G = (V, E)$, a *clique* $G' = (V', E')$ is a subgraph of G, such that for every two vertices in V', there exists an edge $e_i \in E'$ connecting the two vertices. If we consider that nodes are genes and an edge between two nodes exists only when the correlation is higher than a given threshold, a clique will indicate a highly-correlated group of genes, which might belong to the same pathway/functional group.

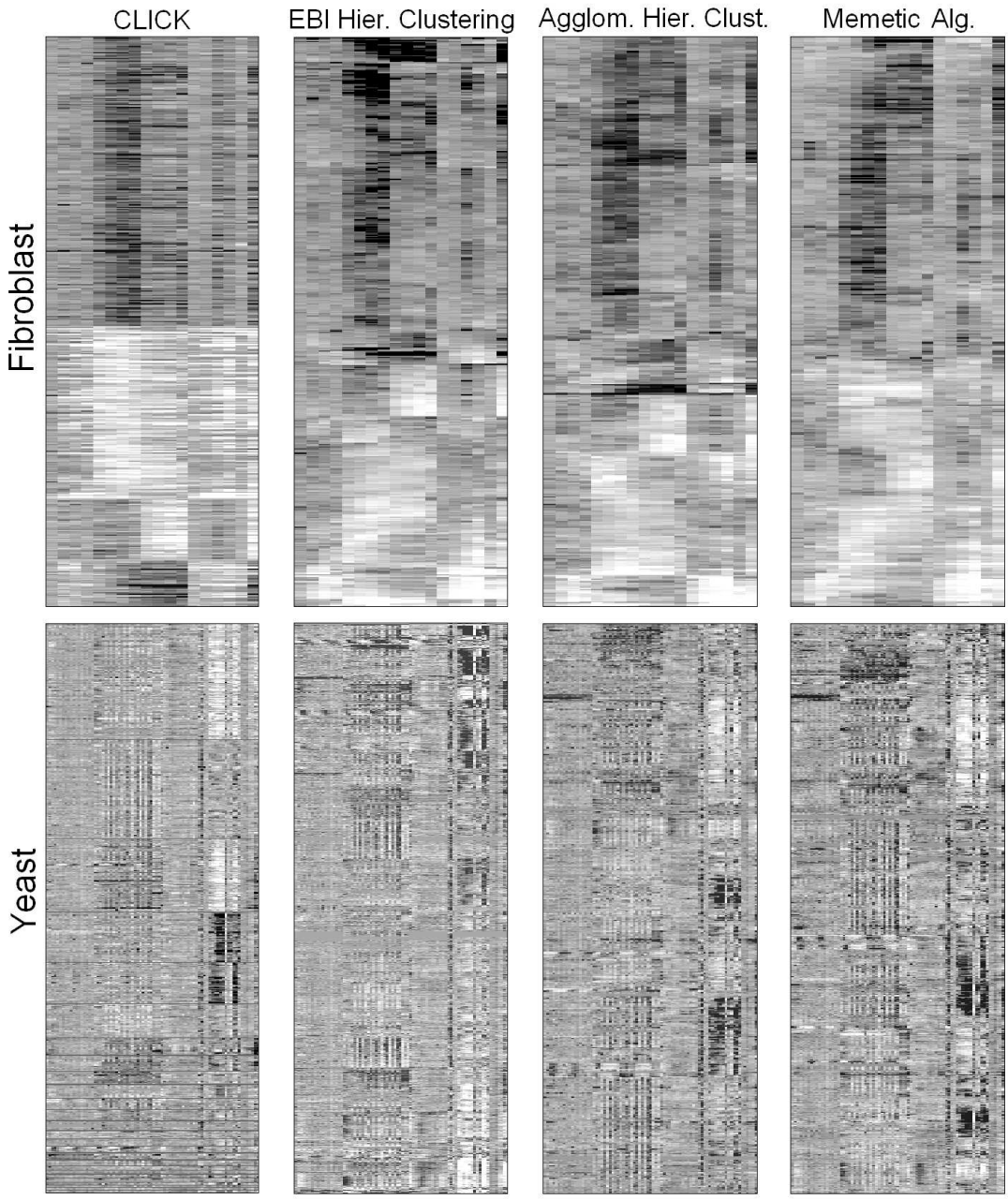

Fig. 15. Results for the *fibroblast* and *yeast* (*Saccharomyces cerevisiae*) instances, with 517 and 6221 genes, respectively.

In order to compare different solutions we used the *average correlation of gene expression patterns at distance d in the ordering* $< \rho > (d)$, which is the average *Pearson* correlation between the genes that are at distance d from each other in the sequence and it is calculated as:

$$< \rho > (d) = \frac{1}{n-d} \sum_{i=1}^{n-d} \rho_{i,i+d}. \tag{3}$$

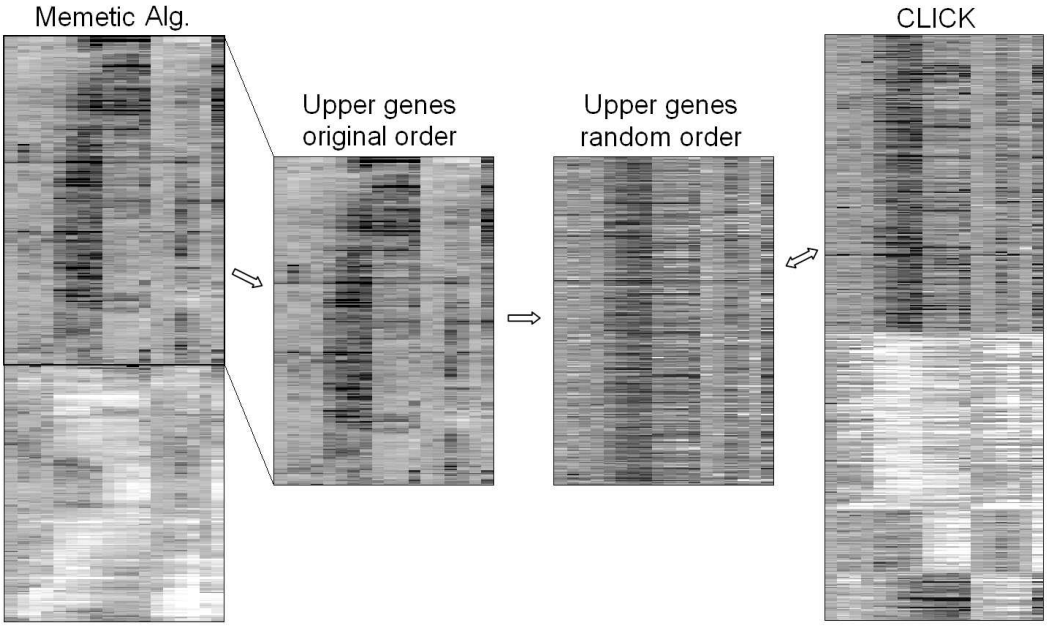

Fig. 16. A comparison of CLICK and MA results on the *fibroblast* instance. CLICK has apparently discovered four major clusters. However, if we randomly order the subsequence given by the MA for the genes of the major cluster, we can see that it is visually similar to the result given by CLICK. This shows that the MA is able to see a rich substructure within that cluster.

In this section we used the same yeast instance described in Section 5.2 introduced by Eisen et al. (1998). In preliminary tests, Eisen's algorithm gave the best results on this instance, so we selected it to make a fair comparison. In order to facilitate reproducibility of our claims by other groups, we filtered out all genes with missing values. That eliminates the need for imputation of missing values and still leaves 3,222 genes with measurements across all the 80 experiments.

In Figures 17 and 18, we show the *average correlation profile* of the solutions obtained by the memetic algorithm and Eisen's hierarchical clustering. The two curves plotted in Figure 17 represent the solutions obtained by our memetic algorithm with two configurations of window size ($w_s = 1$ and $w_s = 5$). The thicker line corresponds to the solution found by Eisen's hierarchical clustering. It is clear that Eisen's method shows a good average correlation for genes that are closer in the sequence, with a relatively soft decay with the increase of the distance. The memetic algorithm, on the other hand, shows profiles with high correlations for genes that are closer in the sequence, but its decay is faster. In Figure 18, the MA's window sizes are larger (1% and 5% of the number of genes). The correlation profiles are quite different from the previous two, with a much slower decay of the curves.

These results show that when we choose small values for the parameter w_s, we obtain smaller groups of correlated genes, but the correlation among them is

high. On the other hand, higher values for w_s are more suitable to find larger functional groups or to get a "whole" picture view of the microarray, as in the Lenna image. However, the correlation among the genes in these large groups is lower.

After counting the number of cliques in the four solutions returned by the memetic algorithm, we found that the configuration $w_s = 5$ was the best for this instance, independent of the correlation threshold. It also consistently surpassed the solution found with Eisen's method in number of cliques (see Figure 19). This indicates a better ordering, with similar genes layout closer than in Eisen's solution. This will be reflected next, when we check the functional groups associated to some of these cliques.

There are many functional groups easily identifiable in both solutions, but *protein synthesis* is by far the largest and easiest to distinguish. In Figure 20, all genes have a very similar activation profile and the two groups contain almost the same genes. Even though the memetic algorithm was able to put three more genes that are related to protein synthesis, we can say that both groups have overall the same quality. The differences start to appear when we move to the smaller, more difficult to find, functional groups. In Figure 21, we show some smaller functional groups found by both methods. They were *protein*

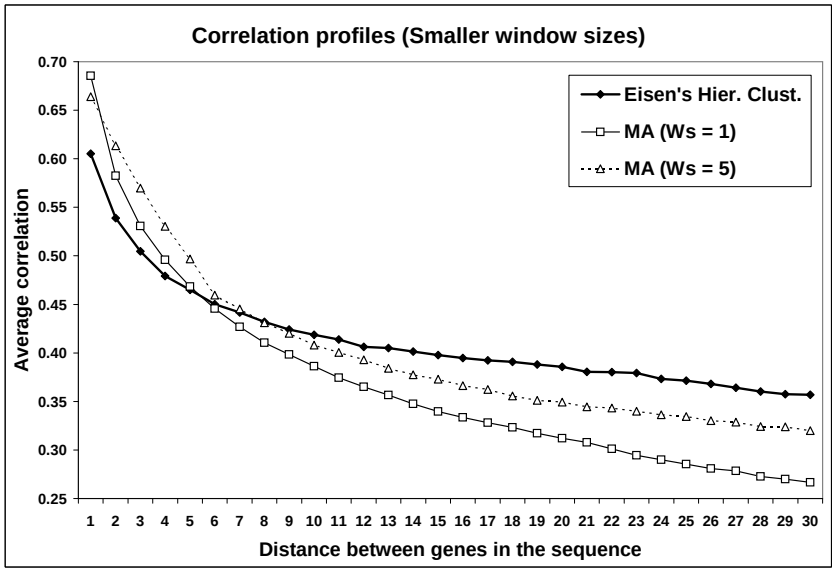

Fig. 17. Comparison between the memetic algorithm results with $w_s = 1$ and $w_s = 5$ and Eisen's hierarchical clustering algorithm for the *Yeast* instance. The lines shown are the *average correlation of gene expression patterns at distance d in the permutation ordering* of the solutions found by the three methods (as a function of d). All of them have the same pattern; genes that are closer in the sequence have a higher correlation than genes that are farther apart. With a small window the average correlations between genes up to five positions apart are higher than in Eisen's solution, making small windows better suited to find small groups of highly-correlated genes.

degradation, *ATP synthesis*, *oxidative phosphorylation*, *TCA cycle* and *sterol metabolism*. The memetic algorithm managed to put together the ATP synthesis, oxidative phosphorylation and TCA cycle functional groups, totalling 15 genes in this group. Eisen's solution divided these genes and put them well apart in the sequence. It is known that ATP synthesis, oxidative phosphorylation and TCA cycle are all related to the respiration process. The MA has put in this group YKR046C, of yet unknown function but for which protein-protein interactions seem to indicate a role in ATP/ADP exchange (Samanta and Liang, 2003; Athenstaedt et al., 1999; Brown et al., 2000). The fact that the memetic algorithm managed to put these groups together seems to reflect their link. For sterol metabolism, the difference was in the number of elements; seven for the memetic algorithm and four for Eisen's. Another TCA cycle functional group was also found by both methods; the memetic one with six genes and Eisen's with only three. It appears that depending on the adjustment of the window size, the memetic algorithm can be very successful in joining groups that are commonly put apart and at the same time continue finding larger functional groups.

These tests indicate that the memetic approach yields a flexibility that other methods lack. If the goal is to get an overall picture of the microarray, one can set the memetic algorithm's window size to a large value. If the aim is to find an order that puts together highly correlated groups, we can reduce the window size and be confident to find such groups.

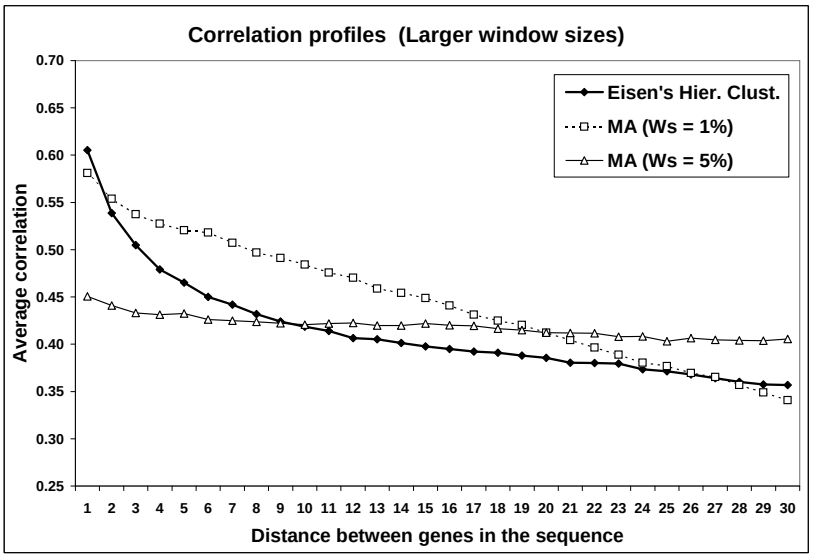

Fig. 18. Comparison between the memetic algorithm results with $w_s = 1\%$ and $w_s = 5\%$ and Eisen's hierarchical clustering algorithm for the *Yeast* instance. The *average correlation of gene expression patterns at distance d in the permutation ordering* $< \rho >$ (d) for the MAs have a very slow decay pattern, meaning that genes that are well-apart in the sequence still have a good correlation. These window sizes are better suited to find larger groups of correlated genes, or to order groups of smaller, highly-correlated clusters.

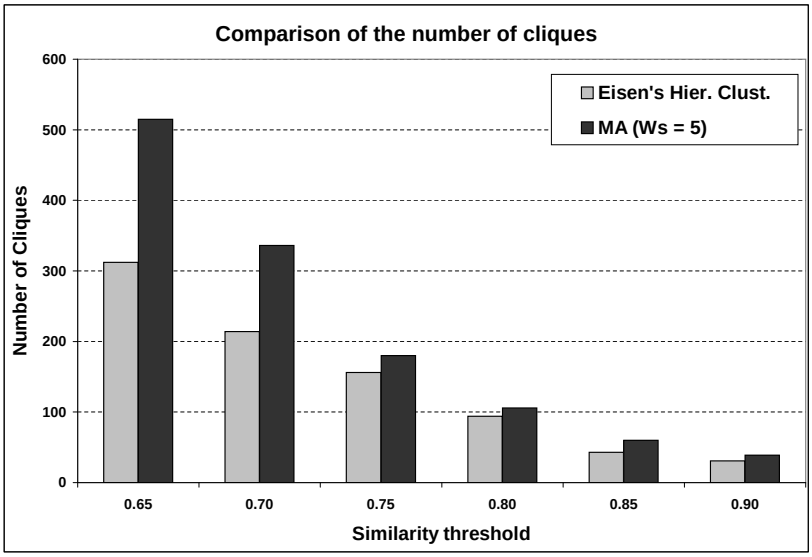

Fig. 19. Total number of cliques that have their genes ordered consecutively in the final display by the MA ($w_s = 5$) and Eisen's hierarchical clustering.

8 Conclusion

This paper presented a new Memetic Algorithm for the Gene Ordering problem. The technique belongs to the Evolutionary Algorithms metaheuristics, and has an embedded Tabu Search optimization procedure. Another important contribution was the introduction of a controlled experimentation to compare different techniques. The quality assessment of a given solution is a critical point, as there is no unique measure for that. Several large images were used as if they were microarray data, and the goal was to arrange their rows, which played the role of genes. In order to emulate the outcome of microarray experiments, different levels of noise were added to the originally error-free image. Our work is a first step towards a more systematic analysis of the impact of microarray data errors in the output and robustness of algorithms.

The *CLICK* algorithm and the *EBI Expression Profiler* hierarchical clustering were used as comparison methods. Although more time consuming, the Memetic Algorithm was able to outperform both of them in most instances. The Memetic Algorithm has also consistently improved the initial solutions provided by our agglomerative hierarchical clustering technique. After validating the new approach with the images, we tested it with two microarray data sets. The first is derived from *Fibroblast* cells, and has 517 genes. The other is the well-known *Yeast* complete microarray, composed of 6,221 genes. Results were displayed as a series of grayscale images, which show the orderings for each algorithm and the high quality results of the Memetic Algorithm proposed in this paper.

The last part of the paper draws a comparison between the memetic algorithm

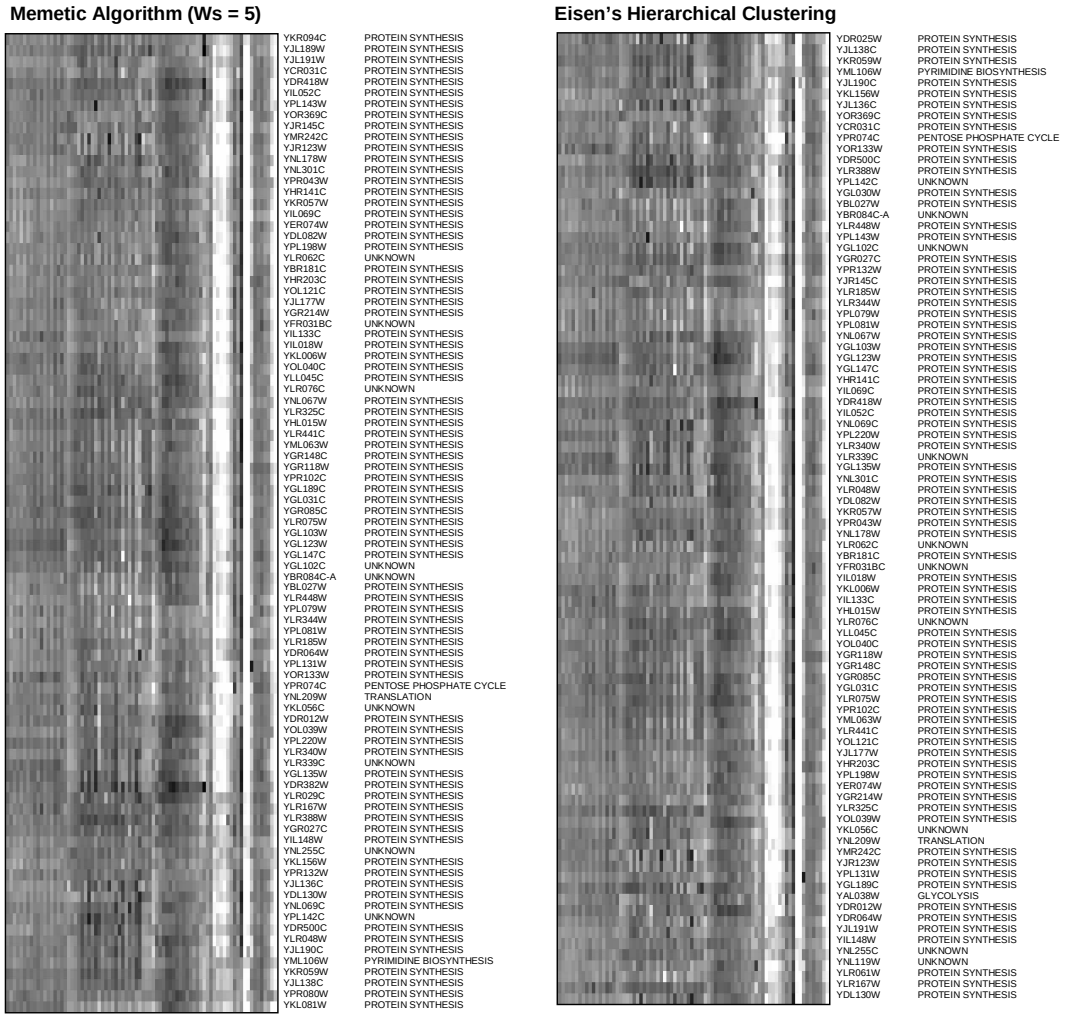

Fig. 20. *Protein synthesis* functional group found by the memetic algorithm and Eisen's hierarchical clustering. The memetic algorithm solution contains 89 genes and 77 of them are *protein synthesis*-related. Eisen's solution contains one gene less and 74 of them are *protein synthesis*-related. Even though larger groups are easy to identify and both methods perform almost the same on them, there generally is a slight difference in favor of the memetic algorithm.

and Eisen's hierarchical clustering, a well-known approach extensively cited in the Bioinformatics and Molecular Biology literature. The conclusion is that the memetic algorithm brings an extra degree of flexibility that other current methods for display lack, allowing the user to adjust it to his/her own needs – a more broad-aimed ordering, where the user can visualize the 'whole picture' of the microarray at hand, or a more detail-aimed ordering, where the smaller functional groups are easier to identify. The ability of the memetic algorithm to find highly-correlated groups of genes, usually related to functional groups or genetic pathways, surpasses the abilities of previous methods.

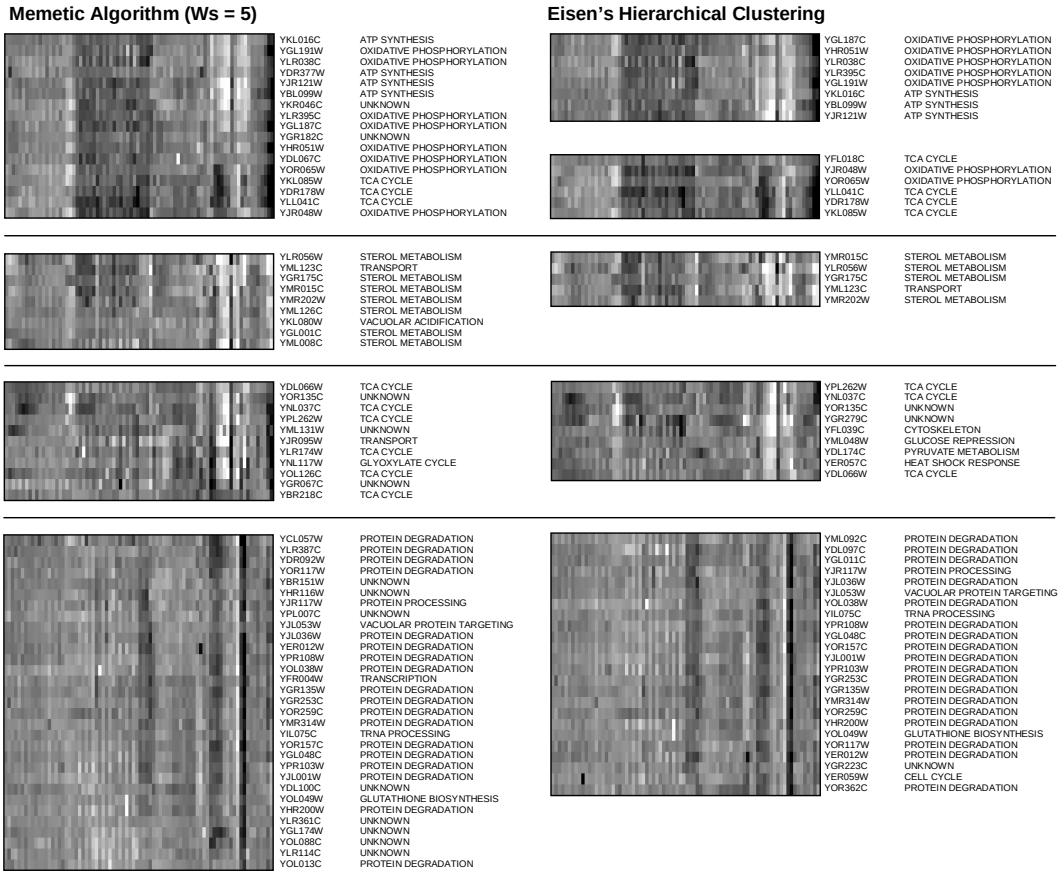

Fig. 21. Smaller functional groups found by the memetic algorithm and Eisen's hierarchical clustering. The first group is *ATP synthesis + oxidative phosphorylation + TCA cycle*, which was separated into two groups well-apart with Eisen's method. For the *sterol metabolism* and *TCA cycle* groups, the memetic algorithm found larger clusters. Finally, both methods grouped the same number of *protein degradation* genes, 18 in total.

Acknowledgements

This work was funded by a Strategic Initiative Fund (SIF) from The University of Newcastle. The authors would also like to thank C. Cotta for several useful discussions.

References

K. Athenstaedt, D. Zweytick, A. Jandrositz, S. D. Kohlwein, and G. Daum. Identification and characterization of major lipid particle proteins of the yeast textitSaccharomyces cerevisiae. *J. Bacteriol.*, 181(20):6441–6448, 1999.

R. Berretta, A. Mendes, and P. Moscato. Integer programming models and algorithms for molecular classification of cancer from microarray data. In

Proceedings of the 28th Australasian Computer Science Conference, pages 361–370, 2005.

R. Berretta and P. Moscato. The number partitioning problem: An open challenge for evolutionary computation? In D. Corne and M. Dorigo, editors, *New Ideas in Optimization*, pages 261–278. McGraw-Hill, 1999.

R. Berretta and L. Rodrigues. A memetic algorithm for multi-stage capacitated lot-sizing problems. *International Journal of Production Economics*, 87(1):67–81, 2004.

T. Biedl, B. Brejova, E. Demaine, A. Hamel, and T. Vinar. Optimal arrangement of leaves in the tree representing hierarchical clustering of gene expression data. Technical Report 2001-14, University of Waterloo, Canada, 2001.

L. Bini, S. Pacini, S. Liberatori, S. Valensin, M. Pellegrini, R. Raggiaschi, V. Pallini, and C. Baldari. Extensive temporally regulated reorganization of the lipid raft proteome following t-cell antigen receptor triggering. *Biochemical Journal*, 369:301–309, 2003.

A. Brazma and J. Vilo. Gene expression data analysis. *Federation of European Biochemical Societies (FEBS) Letters*, 480:17–24, 2000.

M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proceedings of the Natural Academy of Sciences, U.S.A*, 97(1):262–267, 2000.

L. Buriol, P. Franca, and P. Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10(5):483–506, 2004.

C. Cotta, A. Mendes, V. Garcia, P. França, and P. Moscato. Applying memetic algorithms to the analysis of microarray data. In G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, and E. Marchiori, editors, *EvoWorkshops*, volume 2611 of *Lecture Notes in Computer Science*, pages 22–32. Springer, 2003. ISBN 3-540-00976-0.

C. Cotta and J. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18:137–153, 2003.

M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. In *Proceedings of the National Academy of Sciences of the USA 95*, pages 14863–14868, 1998.

R. Elkon, C. Linhart, R. Sharan, R. Shamir, and Y. Shiloh. Genome-wide in silico identification of transcriptional regulators controlling the cell cycle in human cells. *Genome Research*, 13(5):773–780, 2003.

D. Fasulo. An analysis of recent work on clustering algorithms. Technical Report UW-CSEO1-03-02, University of Washington, 1999.

P. Franca, A. Mendes, and P. Moscato. A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132(1):224–242, 2001.

D. Gershon. Microarray technology - an array of opportunities. *Nature*, 416

(6883):885, 2002.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, Massachusetts, 1997.

T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caligiuri, C. Bloomfield, and E. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

K. Gunderson, S. Kruglyak, M. Graige, F. Garcia, B. Kermani, C. Zhao, D. Che, T. Dickinson, E. Wickham, J. Bierle, D. Doucet, M. Milewski, R. Yang, C. Siegmund, J. Haas, L. Zhou, A. Oliphant, J.-B. Fan, S. Barnard, and M. Chee. Decoding randomly ordered DNA arrays. *Genome Research*, 14:870–877, 2004.

V. Iyer et al. The transcriptional program in the response of human fibroblasts to serum. *Science*, 283:83–87, 1999.

G. W. Klau, I. Ljubic, A. Moser, P. Mutzel, P. Neuner, U. P. G. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting steiner tree problem. In *Genetic and Evolutionary Computation - GECCO*, volume 3102, pages 1304–1315. Springer-Verlag, 2004.

R. Kothapalli, S. Yoder, S. Mane, and T. Loughran. Microarray results: How accurate are they? *BMC Bioinformatics*, 3(22):1–10, 2002.

K. Martin, E. Graner, Y. Li, L. Pricea, B. Kritzman, M. Fourniera, E. Rhei, and A. Pardee. High-sensitivity array analysis of gene expression for the early detection of disseminated breast tumor cells in peripheral blood. In *Proceedings of the National Academy of Sciences of the USA 98*, pages 2646–2651, 2001.

P. Merz. Analysis of gene expression profiles: an application of memetic algorithms to the minimum sum-of-squares clustering problem. *BioSystems*, 72:99–109, 2003.

P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

P. Moscato. An introduction to population approaches for optimization and hierarchical objective functions: a discussion on the role of tabu search. *Annals of Operations Research*, 41:85–121, 1993.

P. Moscato, R. Berretta, M. Hourani, A. Mendes, and C. Cotta. Genes related with alzheimer's disease: A comparison of evolutionary search, statistical and integer programming approaches. In F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. D. Smith, and G. Squillero, editors, *EvoWorkshops*, volume 3449 of *Lecture Notes in Computer Science*, pages 84–94. Springer, 2005a. ISBN 3-540-25396-3.

P. Moscato, R. Berretta, and A. Mendes. A new memetic algorithm for ordering datasets: Applications in microarray analysis. In *to appear in Proceed-*

ings of the sixth Metaheuristics International Conference, 2005b.

P. Moscato and M. G. Norman. A 'memetic' approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In M. Valero, E. Onate, M. Jane, J. L. Larriba, and B. Suarez, editors, *Parallel Computing and Transputer Applications*, pages 177–186. IOS Press, Amsterdam, 1992.

S. Pomeroy, P. Tamayo, M. Gaasenbeek, L. Sturla, M. Angelo, M. McLaughlin, J. Kim, L. Goumnerova, P. Black, C. Lau, J. Allen, D. Zagzag, J. Olson, T. Curran, C. Wetmore, J. Biegel, T. Poggio, S. Mukherjee, R. Rifkin, A. Califano, G. Stolovitzky, D. Louis, J. Mesirov, E. Lander, and T. Golub. Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*, 415(6870):436–442, 2002.

J. Puchinger, G. Raidl, and R. M. Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *Proceedings of the sixth Metaheuristics International Conference*, 2005.

W. Raffelsberger, D. Dembl, M. Neubauer, M. Gottardis, and H. Gronemeyer. Quality indicators increase the reliability of microarray data. *Genomics*, 80 (4):385–394, 2002.

M. Samanta and S. Liang. Predicting protein functions from redundancies in large-scale protein interaction networks. *Proceedings of the Natural Academy of Sciences, U.S.A*, 100(22):12579–12583, 2003.

R. Shamir and R. Sharan. Algorithmic approaches to clustering gene expression data. In T. Jiang, T. Smith, Y. Xu, and M. Zhang, editors, *Current Topics in Computational Biology*, pages 269–299. MIT Press, 2002.

R. Sharan and R. Shamir. CLICK: A clustering algorithm with applications to gene expression analysis. In *Proceedings of* 8^{th} *International Conference on Intelligent Systems for Molecular Biology (ISMB'00)*, pages 307–316, 2000.

P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Mol. Biol. Cell*, 9:3273–3297, 1998.

P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub. Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation. In *Proceedings of the National Academy of Sciences of the USA 96*, pages 2907–2912, 1999.

S. Teichmann and M. Babu. Conservation of gene co-regulation in prokaryotes and eukaryotes. *TRENDS in Biotechnology*, 20(10):407–410, 2002.

L. van't Veer, H. Dai, M. van de Vijver, Y. He, A. Hart, M. Mao, H. Peterse, K. van der Kooy, M. Marton, A. Witteveen, G. Schreiber, R. Kerkhoven, C. Roberts, P. Linsley, R. Bernards, and S.H.Friend. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415(6871):530–536, 2002.

J. Vilo, M. Kapushesky, P. Kemmeren, U. Sarkans, and A. Brazma. Expression profiler. In G. Parmigiani, E. Garrett, R. Irizarry, and S. Zeger, editors, *The*

Analysis of Gene Expression Data: Methods and Software. Springer-Verlag, 2003.

E. Wingender, X. Chen, R. Hehl, H. Karas, I. Liebich, V. Matys, T. Meinhardt, M. Pruss, I. Reuter, and F. Schacherer. Transfac: an integrated system for gene expression regulation. *Nucleic Acids Research*, 28:316–319, 2000.

L. Wu, T. Hughes, A. Davierwala, M. Robinson, R. Stoughton, and S. Altschuler. Large-scale prediction of saccharomyces cerevisiae gene function using overlapping transcriptional clusters. *Nature Genetics*, 31(3):255–265, 2002.