BIOINFORMATICS DISCOVERY NOTE

Vol. 1 no. 1 2001
Pages 1 9

Fast Optimal Leaf Ordering for Hierarchical Clustering

Ziv Bar-Joseph, David K. Gifford and Tommi S. Jaakkola

Laboratory for Computer Science, MIT, 545 Technology Square, Cambridge, MA, 02139, USA, Laboratory for Computer Science, MIT, 545 Technology Square, Cambridge, MA, 02139, USA and Artificial Intelligence Laboratory, MIT, 545 Technology Square, Cambridge, MA, 02139, USA

ABSTRACT

We present the first practical algorithm for the optimal linear leaf ordering of trees that are generated by hierarchical clustering. Hierarchical clustering has been extensively used to analyze gene expression data, and we show how optimal leaf ordering can reveal biological structure that is not observed with an existing heuristic ordering method. For a tree with n leaves, there are 2^{n-1} linear orderings consistent with the structure of the tree. Our optimal leaf ordering algorithm runs in time $O(n^4)$, and we present further improvements that make the running time of our algorithm practical.

Contact: zivbj@mit.edu, gifford@lcs.mit.edu, tommi@ai.mit.edu

INTRODUCTION

Hierarchical clustering organizes a set of input elements into a binary tree that groups similar elements together. The distance between input elements in the resulting binary tree is related to their similarity. When input elements are vectors of gene expression levels the resulting tree organizes genes or experiments such that underlying biological structure can often be discerned. Hierarchical clustering has been extensively used in the biological literature to find genes that share a common function (Eisen et al. (1998); Spellman et al. (1998); Alizadeh et al. (2000); Causton et al. (2001)) and for experiment classification (Alon et al. (1999)).

Trees that result from gene clustering are usually displayed with their leaves in a linear order. Biological analysis is often done in the context of this linear ordering, making the linear ordering of the leaves significant. For example, genes or experiments that are adjacent in a linear ordering are often hypothesized to be related in some manner. In addition, when analyzing time series data the entire structure of an ordering may be relied upon to determine which genes are synchronized in time and to reference the timing of gene clusters to one another.

An *optimal leaf ordering* of a binary tree maximizes the sum of the similarity of adjacent elements in the ordering. For a binary tree of n input elements there are 2^{n-1} linear orderings consistent with the structure of the tree. Because of the large number of distinct linear orderings finding an optimal leaf ordering was thought to be impractical (Eisen et al. (1998)). Our primary results are the demonstration of a practical algorithm for determining optimal orderings, and showing that optimal leaf orderings are useful.

Hierarchical clustering

Hierarchical clustering starts by computing a similarity matrix (S) between all pairs of input elements to be clustered. Initially, each input element corresponds to a single cluster. Using the similarity matrix S we combine at each step the two most similar clusters and form a new cluster which contains both clusters. After combining these two clusters we compute the similarity of the new cluster to all the remaining clusters. For n genes, this step is repeated $n-1$ times until we are left with a single cluster that contains all genes. Eisen et al. (1998) contains a detailed description of the hierarchical clustering algorithm. The running time of this algorithm is $O(n^3)$ since for each of the $n-1$ mergings we perform we need to search the S matrix for the largest entry. Though hierarchical clustering could be performed in $O(n^2)$ as described in Eppstein (1998), the above algorithm is the one that is implemented in Cluster, the software package described in Eisen et al. (1998), and is the one most papers use.

Unlike most other clustering methods, hierarchical clustering does not determine unique clusters. Thus, the user has to determine which of the subtrees are clusters, and which subtrees are only a part of a bigger cluster. Any improvement to the basic algorithm, such as a leaf ordering algorithm, could help a user identify clusters and interpret the data. In addition, one might be interested not only in the clusters but also in their relationships. Cluster relationships are key when studying time-series data.

Current versions of hierarchical clustering software use heuristics for leaf ordering. Cluster (Eisen et al. (1998)) orders leaves based on their average expression level. A

second heuristic for leaf ordering is suggested by the Cluster manual, and orders clusters using the results of a one dimensional self organizing map. Because self-organizing maps and hierarchical clustering use different methods to perform clustering (see Tamayo et al. (1999)), clusters generated by the self-organizing map algorithm do not always match those generated by the hierarchical clustering algorithm. In addition, self-organizing maps only order the clusters, and does not order genes inside each cluster. A third heuristic, which was presented by Alon et al. (1999), orders leaves and internal nodes based on their similarity to their parent's siblings. This heuristic uses local similarities to generate the global ordering, and usually does not lead to optimal ordering.

Results

We show that it is practical to compute the optimal leaf ordering for a given hierarchical clustering tree. Our algorithm for the optimal leaf ordering for a tree of n leaves runs in $O(n^4)$ time and $O(n^2)$ space. Our method is general and works for any binary tree leaf ordering and thus is not restricted to trees that were generated using hierarchical clustering. Because hierarchical clustering is performed in O(n^3) time and is thus faster than our algorithm for optimal leaf ordering, further improvements were necessary to make our algorithm practical. These improvements make our algorithm's running time very reasonable when compared with tree construction time.

We present several examples that show that optimal leaf ordering achieves results that are superior to the heuristic ordering method that is present in the existing software package of Eisen et al. (1998). These examples consist of random and hand generated data as well as biological data. For example, we tested our algorithm on the cell cycle data of Spellman et al. (1998). Using optimal ordering, we where able to uncover not only the correct cell cycle specific clusters, but also properly order these clusters in time.

LEAF ORDERING ALGORITHM

In this section we formalize the optimal ordering problem. We then present our leaf ordering algorithm, and analyze its time and space complexity.

Problem definition

First, we formalize the optimal leaf ordering problem, using the following notations. For a tree T with n leaves, denote by $z_1, \cdots, z_n$ the leaves of T and by $v_1 \cdots v_{n-1}$ the $n-1$ internal nodes of T. A **linear ordering consistent with** T is defined to be an ordering of the leaves of T generated by flipping internal nodes in T (that is, changing the order between the two subtrees rooted at v_i, for any $v_i \in T$). See Figure 1 for an example of node flipping. Since there are $n-1$ internal nodes, there are 2^{n-1} possible linear orderings of the leaves of the tree. Our goal is to find an ordering of the tree leaves, that maximizes the sum of the similarities of adjacent leaves in the ordering. This could be stated mathematically in the following way. Denote by Φ the space of the 2^{n-1} possible orderings of the tree leaves. For $\phi \in \Phi$ we define $D^{\phi}(T)$ to be:

$$D^{\phi}(T) = \sum_{i=1}^{n-1} S(z_{\phi_i}, z_{\phi_{i+1}})$$

where z_{ϕ_i} is the ith leaf when T is ordered according to ϕ, and S is the similarity matrix. Thus, our goal is to find an ordering ϕ that maximize $D^{\phi}(T)$. For such an ordering ϕ, we say that $D(T) = D^{\phi}(T)$.

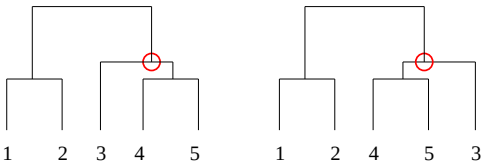


Fig. 1. Change in the leaf ordering due to an internal node flip. When flipping the two subtrees rooted at the red circled node we obtain different orderings while maintaining the same tree structure. Since there are $n-1$ internal nodes there are 2^{n-1} possible orderings of the tree leaves.

Algorithm

When describing our algorithm we refer to an internal node, and a subtree rooted at that internal node using the same name. A leaf z is said to belong to a subtree v if z is one of the leaves of v. For a subtree v, $|v|$ denotes the number of leaves in v.

The algorithm we present is recursive, and works in a way that resembles dynamic programming. For each internal node v it finds the cost of the optimal ordering of the subtree rooted at v, denoted by $M(v)$. In addition, denote by v_l and v_r the two internal nodes just underneath v (i.e. the nodes for which v is their parent node, see Figure 2). For every pair of leaves $u \in v_l$ and $w \in v_r$, our algorithm computes $M(v, u, w)$ which is the optimal linear ordering of v when the leftmost leaf of v is u and the rightmost leaf of v is w.

Our algorithm works in a bottom up way. No M value for a subtree v is computed until all the M values of all subtrees of v are computed. Thus, when computing $M(v, u, w)$ for a subtree v, we already computed the M values for the two subtrees of v, v_l and v_r. If we now specify that some leaf m will be the rightmost leaf of v_l and a leaf k will be the leftmost leaf of v_r (see Figure 2) then the best ordering of v when u is on one side, w

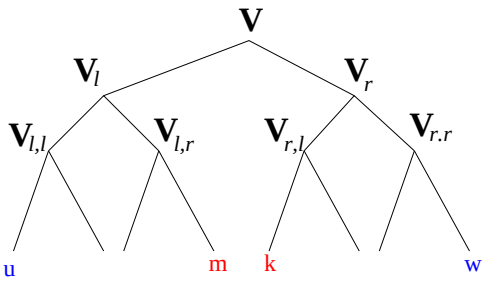


Fig. 2. For every pair of leaves $u \in v_l$ and $w \in v_r$ our algorithm computes $M(v,u,w)$ which is the optimal linear ordering when the leftmost leaf of v is u and the rightmost leaf of v is w. Note that when computing $M(v,u,w)$ we must have a leaf $m \in v_{l,r}$ as the rightmost leaf of v_l and $k \in v_{r,l}$ as the leftmost leaf of v_r.

is on the other and k and m are in the middle is just: $M(v_l,u,m) + M(v_r,w,k) + S(m,k)$ where S is the similarity matrix. In order to find $M(v,u,w)$ we must try all feasible m,k pairs, compute the resulting value, and take the one that has the highest score. This could be summarized by the following equation: $M(v,u,w) = max_{m \in v_{l,r}, k \in v_{r,l}} M(v_l,u,m) + M(v_r,w,k) + S(m,k)$, where $v_{l,r}$ is the right subtree of v_l and $v_{r,l}$ is the left subtree of v_r (see Figure 2). Since in every linear ordering of the leaves of v there must be a leaf from v_l on the left side of v and a leaf from v_r on the right side of v (or vice versa), once we computed $M(v,u,w)$ for all possible pairs u,w, we just need to find the highest scoring pair in order to find $M(v)$. Thus, if v_t is the root of the input tree T, we have $D(T) = M(v_t)$.

After computing $M(v_t)$ we use backtracking to find the path we took in order to arrive at $M(v_t)$. This gives the actual ordering of the leaves of T. This can be done in a similar way to the backtracking used in dynamic programming.

The algorithm for computing $M(v_t,L,R)$ (where L,R stands for all the possible leftmost and rightmost pairs of leaves of v) is presented in Figure 3. As mentioned above, finding the maximum over $M(v_t,L,R)$ results in $M(v_t)$. A complete formal proof of the correctness of the algorithm can be found on our website (Bar-Joseph et al. (2001)).

Algorithm complexity

The time complexity of our algorithm is $O(n^4)$. This can be seen by observing the following property of our algorithm. For each pair of leaves (u,w) there exists one, and only one subtree v for which we compute $M(v,u,w)$. This is the subtree rooted at the least common ancestor of u and w. For example, in Figure 2, v is the least common ancestor of the pair (u,w) while v_l is the least common ancestor of the pair (u,m). If v is the least common ancestor of (u,w) then u could be on one side of v while w is on the other. However, if v is not the least common ancestor of u and w and $u,w \in v$, it means that u and w belong to the same subtree of v (either v_l or v_r). Thus u and w cannot appear on different sides in an ordering of the leaves of v and we do not compute $M(v,u,w)$ for v.

Thus, we only compute $M(v,u,w)$ once for each of the $O(n^2)$ pairs of leaves. Each computation of $M(v,u,w)$ requires us to find the maximum over all the possible m,k leaves that lie in the intersection of v_l and v_r. This results in at most $O(n^2)$ time needed to compute each $M(v,u,w)$. Thus, the total running time of our algorithm is at most $O(n^4)$. It could be shown that this is the actual running time of our algorithm on a complete balanced binary tree [†]. A complete proof of the time complexity can be found on our website (Bar-Joseph et al. (2001)).

The space complexity of our algorithm is $O(n^2)$. As shown above, for every pair of leaves u,w we compute $M(v,u,w)$ only once, and thus we store only one value for each such pair. In order to perform the backtracking needed to reconstruct the order of the leaves of the tree we only need to keep two pointers for each $M(v,u,w)$ entry (one points to $M(v_l,u,m)$ and the other to $M(v_r,w,k)$). Thus, the space complexity of our algorithm is $O(n^2)$. Note that since the size of the similarity matrix S is also $O(n^2)$, $O(n^2)$ is the optimal space complexity if we wish to compute S.

IMPROVING THE ORIGINAL ALGORITHM

In this section we discuss how we can improve the running time of our algorithm without sacrificing optimal linear ordering. For microarray expression data, n is in many cases bigger than 1000. Thus, an $O(n^4)$ algorithm can take a very long time in comparison to a $O(n^3)$ tree construction time. For example, for $n = 1500$ the hierarchical clustering algorithm takes on average 2 minutes while the leaf ordering can take up to 7 hours (see Figure 5).

Early Termination of the search

As discussed in the previous section, for each $M(v,u,w)$ computation we perform an $O(n^2)$ search on all the possible m,k pairs of leaves. Our goal is to terminate this search early when it can not lead to an improvement of $M(v,u,w)$ reducing the computation time. To this end we first order $M(v_l,u,R)$ and $M(v_r,w,L)$ in descending order (where R denotes the set of possible rightmost leaves of v_l when u is the leftmost leaf of v_l, and R is the same for w and v_r).

[†] A balanced binary tree is a tree in which for every subtree v, $|v_l| = |v_r| = |v|/2$.

```
optOrdering(v, S) {
    If |v| = 1 {                    // v has only one leaf
      M(v, u, u) = 0                // u is the only leaf in v
      return M(v, u, u)
    Else                            // v has more than one leaf
      M(v_l, L, R) = optOrdering(v_l, S)      // v_l is the left subtree of v
      M(v_r, L, R) = optOrdering(v_r, S)      // v_r is the right subtree of v
      For all leaves u ∈ v_l {
        For all leaves w ∈ v_r {
          M(v, u, w) = max_{m∈v_{l,r}, k∈v_{r,l}} M(v_l, u, m) + M(v_r, w, k) + S(m, k)
          M(v, w, u) = M(v, u, w)
        }
      }
      return M(v, L, R)             // L, R stands for all possible pairs of leaves from v_l and v_r
}
```

Fig. 3. The recursive algorithm for computing $M(v, L, R)$. This algorithm returns the optimal ordering of v for all pairs of leaves $u \in v_l$ and $w \in v_r$.

Next we find the maximal similarity between a leaf of v_l and a leaf of v_r. Denote this similarity by $C(v_l, v_r) = max_{m \in v_l, k \in v_r} S(m, k)$. When computing $M(v, u, w)$ we go over all possible intersecting leaves m, k according to the order specified above. For each such pair we know that $S(m, k) \leq C(v_l, v_r)$. This can help us terminate the search for $M(v, u, w)$ in the following way.

For each pair of leaves $u \in v_l$, $w \in v_r$ our improved algorithm computes $M(v, u, w)$ by performing two loops (see Figure 4). The outer loop runs over all possible leaves $m \in v_{l,r}$ according to the order of $M(v_l, u, R)$ and the inner loop is performed according to the order of $M(v_r, w, L)$. Denote by $curMax$ the current maximum we have for $M(v, u, w)$. If for a given pair of leaves m, k we have $M(v_l, u, m) + M(v_r, w, k) + C(v_l, v_r) \leq curMax$, we are guaranteed that for any $k' > k$ (that is, k' that comes after k in the ordering of $M(v_r, w, L)$), m, k' will not lead to better value of $M(v, u, w)$. For such a k', $M(v_r, w, k') \leq M(v_r, w, k)$ and $S(m, k') \leq C(v_l, v_r)$ and so $M(v_l, u, m) + M(v_r, w, k') + S(m, k') \leq M(v_l, u, m) + M(v_r, w, k) + C(v_l, v_r) \leq curMax$. Thus, in this case we can terminate the inner loop and jump to the next leaf in the outer loop. The same argument shows that we can terminate the outer loop (and thus terminate the search for $M(v, u, w)$) if for k_0, the leaf that maximizes $M(v_r, w, R)$, (k_0 is the first leaf in the ordering of $M(v_r, w, L)$) we have $M(v_l, u, m) + M(v_r, w, k_0) + C(v_l, v_r) \leq curMax$.

The final part of our improvement is achieved by replacing $C(v_l, v_r)$ with another value for each possible combination of the four subtrees $v_{l,l}, v_{l,r}, v_{r,l}$ and $v_{r,r}$. When perfroming the search for $M(v, u, w)$ where $u \in v_{l,l}$ and $w \in v_{r,r}$ we can replace $C(v_l, v_r)$ with $C(v_{l,r}, v_{r,l})$ since the m, k leaves must come from $v_{l,r}$ and $v_{r,l}$ respectively (see Figure 2).

Though the worst case running time reamains $O(n^4)$, this improvement dramatically decreases the computation time on average, as can be seen in Figure 5. The results of Figure 5 were obtained using randomly generated data. Clustering randomly generated data usually results in a balanced binary tree. As mentioned earlier, balanced binary trees are the worst case for our algorithm and the less balanced the tree is, the faster the computation time. For biological data, the trees generated are usually less balanced when compared with randomly generated trees. In addition, in biological datasets, $C(v_l, v_r)$ is usually smaller between clusters than it is between clusters of the same size for randomly generated datasets. Thus, the actual running time of our algorithm on biological datasets is usually faster when compared with randomly generated datasets with the same number of leaves. The running time of our algorithm on a number of biological datasets appears in Figure 6.

RESULTS

We tested our algorithm on several different inputs. First, we used randomly generated datasets to test the effect that optimal ordering has on the sum of similarities of neighboring leaves, which is the function our algorithm maximizes. Next, we generated datasets to test the effect optimal ordering has on the visual representation of the hierarchical clustering results. We conclude this section by presenting biological results that were obtained using the cell cycle data of Spellman et al. (1998).

For computing the hierarchical clustering results we

$curMax = -\infty$
For all leaves m according to the order of $M(v_l, u, R)$ {
  if $M(v_l, u, m) + M(v_r, w, k_0) + C(v_l, v_r) \leq curMax$ // k_0 is first in the order of $M(v_r, w, L)$
    $M(v, u, w) = curMax$, terminate the search
  For all leaves k according to the order of $M(v_r, w, L)$
    if $M(v_l, u, m) + M(v_r, w, k) + C(v_l, v_r) \leq curMax$
      break // terminate the inner loop
    if $curMax < M(v_l, u, m) + M(v_r, w, k) + S(m, k)$
      $curMax = M(v_l, u, m) + M(v_r, w, k) + S(m, k)$
  } // end of the inner loop
} // end of the outer loop
$M(v, u, w) = curMax$

Fig. 4. The improved way to compute $M(v, u, w)$. This repalces the search over all leaves m, k in the algorithm presented in the previous section.

Num. of genes	Clustering time	Ordering time (original)	Ordering time (with improvement)
100	0	0	0
300	1	22	2
500	3	200	10
1000	32	4000 (66 min)	102
1500	115	24800 (7 hours)	420

Fig. 5. A comparison between the clustering time (in seconds) and the ordering time on randomly generated data. For each leaf we generated 60 data points at random and then computed the resulting hierarchical clustering tree. As can be seen, without the suggested improvements the ordering time can be very slow. However, with the improvements the ordering time is very reasonable when compared with the clustering time. The ordering time for biological datasets is usually faster than that of randomly generated data as explained in the text. These results were obtained on a 400 MHz Pentium pc with 128M memory.

used the correlation coefficients as the similarity matrix (S). The clustering was performed using the average linkage method as presented in Eisen et al. (1998).

Random data

For random data, we chose 60 values (representing 60 time points) at random for each leaf. Next, we computed the resulting similarity matrix, and then hierarchically clustered these data points. Denote by $S(T^r)$ and by $D(T^r)$ the sum of the similarity between adjacent leaves in the initial and optimal leaf ordering of the resulting tree T^r respectively. Set $I = (D(T^r) - S(T^r))/S(T^r)$. I is the increase in similarity of $D(T^r)$ compared with $S(T^r)$. We found that even for a large number of random data points ($n = 1500$), I is on average 20%, indicating that optimal ordering has a noticeable impact on the similarity of neighboring leaves in the linear ordering.

Artificial data

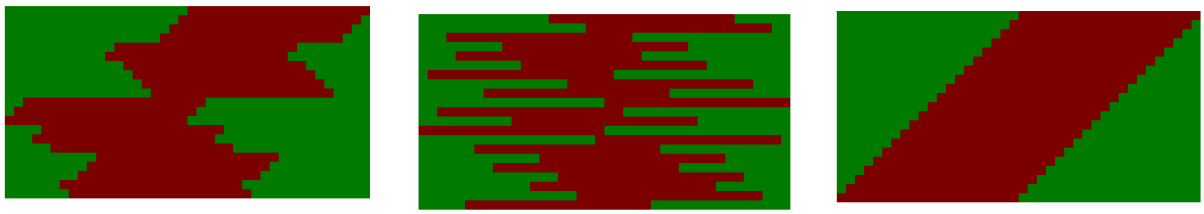
In order to test the effect of leaf ordering on the presentation of the data, we generated several input data sets. Each data set had some structure which we permuted, and then we ran the hierarchical clustering algorithm to cluster the data set. In this section we compare the results of Cluster (Eisen et al. (1998)) with our optimal ordering algorithm. All the figures in this section were generated using TreeView (Eisen et al. (1998)).

In Figure 7 we present structured synthetic data. As can be seen, although there is some structure in the Cluster output, it is obvious that the optimal ordering algorithm captures the true underline structure.

Our next experiment was a larger scale test. We gen-

Type of dataset	Num of experiments	Num of genes	Clustering time	Ordering time
Cell cycle from Spellman et al. (1998)	59	800	15	27
Cell cycle - cdc15 from Spellman et al. (1998)	24	800	15	180
Different sources from Eisen et al. (1998)	79	979	26	50
Environment response from Causton et al. (2001)	45	3684	910 (15 min)	257 (4 min)

Fig. 6. Comparison between the clustering time (in seconds) and the ordering time on a number of biological datasets. The results for the Environment response dataset were obtained on a 700MHz, 512M Pentium pc. The rest were obtained on a 400MHz, 128M Pentium pc. As can be seen, in most cases the ordering time is very reasonable compared to the clustering time. For the largest dataset, the ordering time was actually much smaller than the clustering time.



Cluster result Input (permuted) Ordering result

Fig. 7. Comparison between the ordered and non ordered hierarchical clustering on a generated dataset. Green corresponds to decrease in value (-1) and red to increase (1).

erated 100 data points for each of the 1000 leaves. For each leaf we set 40 consecutive points to -1, and then flip them to 0 or 1 with low probability. The rest of the points were generated at random. The results of the initial ordering (which is similar to the random ordering of Cluster), and of the similarity maximizing ordering are shown in Figure 8. As can be seen, the clusters were identified correctly by the hierarchical clustering algorithm. However, the ordering of the clusters is much better when using optimal leaf ordering.

Biological datasets

We further tested our algorithm using 800 genes which are cell cycle regulated in $Saccharomyces\ cerevisiae$. Spellman et al. (1998) assigned these 800 genes to five groups termed $G1$, S, $S/G2$, $G2/M$, and $M/G1$. These groups approximate the commonly used cell cycle groups in the literature. The assignment to different groups was performed by what the authors call a 'phasing' method. This method compares the 'peak expression' for each unknown gene with the expression of genes that were known to belong to each of these group. Genes were assigned to the group to which their peak expression was the most similar.

After performing the phasing algorithm the authors organized the 800 genes using hierarchical clustering and a heuristic leaf ordering. Most of the genes that belong to the same group appeared together. However, some clusters were a mix of two or more groups, and the cell cycle genes were not presented in temporal order.

Using a subset of the dataset from Spellman et al. (1998) (24 cdc15 experiments), we compared the results of our optimal ordering algorithm to the heuristic ordering in Cluster. As can be seen in Figure 9, our algorithm was able to correctly recover the cell cycle order ($G2/M$, $M/G1$, $G1$, S and $S/G2$). In addition, our algorithm was able to reorder some of the clusters so that the different clusters are correctly separated. For example, the $G1$, S and $S/G2$ groups are mixed in the hierarchical clustering results but are correctly separated in the optimal ordering results. Thus, while still using unsupervised learning, our algorithm was able to correctly identify the cell cycle groups and the order of these groups, achieving a high correlation with the phasing method (which is a supervised algorithm) that was previously used in Spellman et al. (1998).

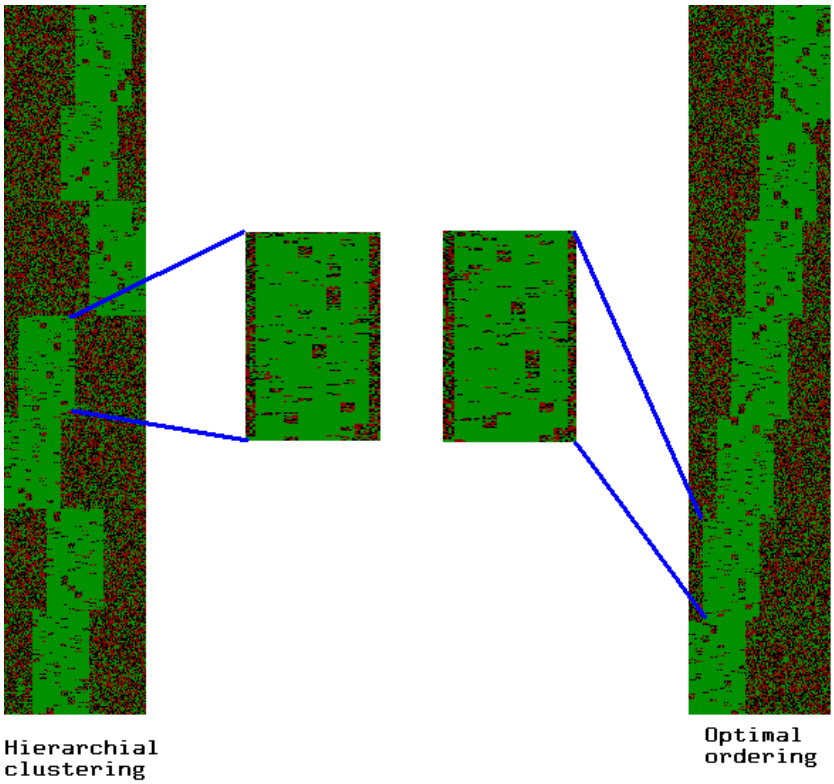


Fig. 8. Comparison between the ordered and non ordered hierarchical clustering. The small images are enlargements of the same cluster in both results. As can be seen, the ordering algorithm orders the clusters so that the relationship between them is apparent. In addition, it also orders the genes inside each cluster as can be seen in the small images.

We also performed a larger scale comparison between the heuristic ordering in Cluster and our optimal ordering algorithm. For this comparison we used the complete dataset of Spellman et al. (1998). This dataset is a combination of three different experiments (cdc15, cdc28 and α factor). These three datasets consisted of 59 separate experiments. We compared the clustering of the 800 genes that were identified as cell cycle regulated. Again, our optimal ordering algorithm was able to recover the order of the different cell cycle groups (unlike the hierarchical clustering), and did a better job in reconstructing the groups themselves. See our website (Bar-Joseph et al. (2001)) for the comparison figures and complete data.

DISCUSSION AND FUTURE WORK

We showed that an optimal linear ordering can be computed in a $O(n^4)$ time. Our algorithm is general, and works for any binary tree leaf ordering. Improving the algorithm by early termination of search paths decreases the average running time of the algorithm dramatically. We intend to make the software used for generating the results in this paper publicly available.

Using optimal ordering one can arrive at results that are better than a heuristic ordering of the leaves. We presented several examples in which the results of our optimal ordering algorithm are superior to the original heuristic ordering results. Optimal leaf ordering helps a user determine meaningful cluster boundaries and also helps determine the relationship between different clusters. These relationships are very important in time series data analysis.

Optimal leaf ordering causes input elements that are highly correlated in a cluster to appear in the middle of the linear ordering of the cluster, while marginally related elements are on the borders of the cluster. We have examined other datasets from Eisen et al. (1998) using the MIPS yeast complexes database (http://www.mips.biochem.mpg.de/

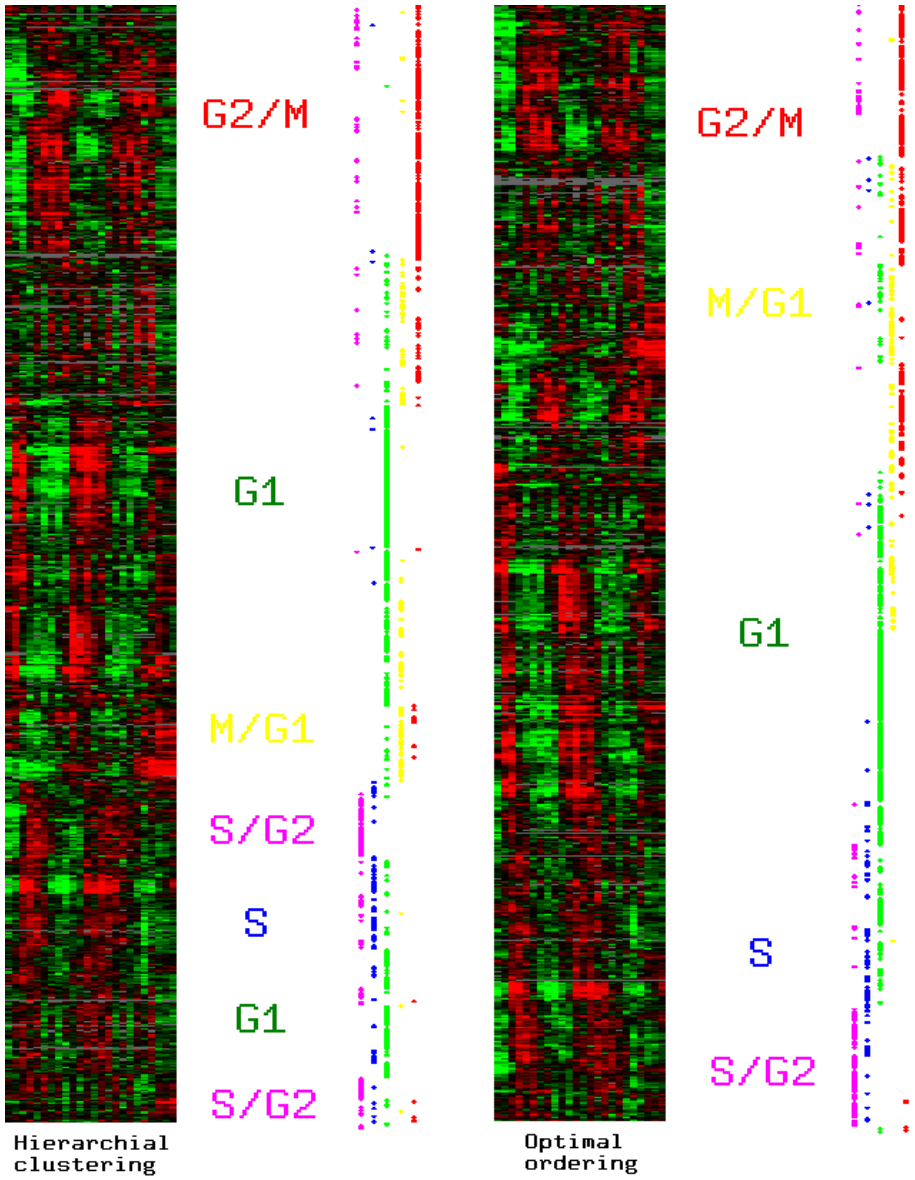


Fig. 9. Color comparison of hierarchical clustering (left) and optimal ordering (right) using the cell cycle data of Spellman et al. (1998). For each of the genes in these figures we plotted a dot on its right which represents the group to which it belongs according to Spellman et al. (1998) (red for $G2/M$ yellow for $M/G1$ etc.). As can be seen , the cell cycle phasing is much more apparent in the optimal ordering result,which correctly recovers the order of the groups in the cell cycle. In addition, using optimal ordering one can better reconstruct the groups themselves as can be seen in the case of the $G1$, S and $S/G2$ groups.

proj/yeast/catalogues/complexes). We have determined that under optimal linear ordering the centers of clusters have common MIPS categorizations, while under Cluster's ordering genes with common MIPS categories were more dispersed. See our website (Bar-Joseph et al. (2001)) for quantitative results and figures.

We are investigating ways of using optimal leaf ordering for automatic cluster discovery. We expect that the structure of an optimal ordering can be employed to determine clusters of varying sizes that are related at desired significance levels.

ACKNOWLEDGEMENTS

We would like to thank Peter Clote for useful comments and discussions, and Fritz Roth for pointing us to earlier work. We are grateful to Dan Gusfield for helping us revise the description of the algorithm, making it much clearer than before.

REFERENCES

Alizadeh, A., M. Eisen, and et al (2000). Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature 403*, 503–510.

Alon, U., N. Barkai, and et al (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. Natl. Acad. Sci. USA 96*, 6745–6750.

Bar-Joseph, Z., D. Gifford, and T. Jaakkola (2001). Fast optimal leaf ordering for hierarchical clustering. *http://www.psrg.lcs.mit.edu/ zivbj/ismb01/optimal.html*.

Causton, H. C., B. Ren, and et al (2001). Remodeling of yeast genome expression in response to environmental changes. *Mol. Biol. of the Cell 12*, 323–337.

Eisen, M., P. Spellman, P. Brown, and D. Botstein (1998). Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA 95*, 14863–14868.

Eppstein, D. (1998). Fast hierarchical clustering and other applications of dynamic closest pairs. In *Proc. of the 9th ACM-SIAM Symp. on Discrete Algorithms*, pp. 619–628.

Spellman, T. S., G. Sherlock, and et al (1998). Comprehensive identification of cell cycle-regulated genes of the yeast $saccharomyces\ cerevisia$ by microarray hybridization. *Mol. Biol. of the Cell 9*, 3273–3297.

Tamayo, P., D. Slonim, and et al (1999). Interpreting patterns of gene expression with self organizing maps: Methods and applications to hematopoietic differentiation. *Proc. Natl. Acad. Sci. USA 96*, 2907–2912.