Supplementary Information to accompany Reverse Engineering Dynamic Temporal Models of Biological Processes and their Relationships

N. Ramakrishnan *et al.*

This document provides supplementary details to accompany the manuscript, in particular it presents a complete description of the algorithm and an outline of our experimental methodology. Complete Gantt charts for each of the stresses analyzed is also presented here.

1 Algorithm Description

As discussed in the main text, the 'unit' of our analysis is to simultaneously cluster two neighboring windows such that the clusters are maximally dissimilar in terms of the related samples assigned to them. This unit is then composed to yield a tiling of the entire time series, or a segmentation.

1.1 Formulation

Given a gene vector $\mathbf{g}_k$, let its projection onto the 'left' window $w_{t_a}^{t_b}$ be referred to as $\mathbf{x}_k$, and its projection onto the 'right' window $w_{t_{b+1}}^{t_c}$ be referred to as $\mathbf{y}_k$. Recall that sets of such projections are clustered separately such that the clusters are maximally dissimilar. Let r and c be the number of clusters for $\mathbf{x}$ and $\mathbf{y}$ vectors, which results in an $r \times c$ contingency table. Let $\mathbf{m}_i^{(x)}$ be the prototype vector for the i^{th} cluster of the $\mathbf{x}$ vectors. The random variable $V^{(\mathbf{x}_k)}$ denotes the assignment of the data vector $\mathbf{x}_k$ to the clusters: the probability of $\mathbf{x}_k$ being assigned to cluster i is given by $P(V^{(\mathbf{x}_k)} = i) = v_i^{(\mathbf{x}_k)}$, where $\sum_{i=1}^{r} v_i^{(\mathbf{x}_k)} = 1$. We refer to the probabilities $v_i^{(\mathbf{x}_k)}$ as cluster membership indicator variables. Similar cluster prototypes $\mathbf{m}_j^{(y)}$, random variables $V^{(\mathbf{y}_k)}$, and cluster indicator variables $v_j^{(\mathbf{y}_k)}$ are defined for $\mathbf{y}$ vectors as well. We denote the probability mass function associated with each $V^{(\mathbf{x}_k)}$ as $p_{V^{(\mathbf{x}_k)}}$, and with each $V^{(\mathbf{y}_k)}$ as $p_{V^{(\mathbf{y}_k)}}$. Then the contingency table counts can be calculated as $n_{ij} = \sum_{k=1}^{N} v_i^{(\mathbf{x}_k)} v_j^{(\mathbf{y}_k)}$. In *hard clustering* algorithms, like the traditional k-means, each data sample is assigned to the nearest cluster with a probability of 1. However, calculating n_{ij} using hard memberships renders the function $\mathcal{F}$ (see main text) nondifferentiable at certain points, as a result of which, we cannot leverage classical numerical optimization algorithms to minimize $\mathcal{F}$. To avoid this problem, cluster indicator variables are typically treated as continuous real variables making $\mathcal{F}$ a smooth function that is continuously differentiable and assigning a nonzero cluster membership probability for each data sample, i.e, $v_i^{(\mathbf{x}_k)}, v_j^{(\mathbf{y}_k)} \in (0, 1)$.

1.2 Smoothing Cluster Probabilities

There are many ways of smoothing, one approach being the use of a Gaussian kernel between the vector and the cluster prototype. We present a novel derivation of this kernel that explains how the error in the

smoothing can be explicitly controlled and also suggests other formulations for smoothing. First, we define

$$\gamma_{(i,i')}(\mathbf{x}_k) = \frac{||\mathbf{x}_k - \mathbf{m}_{i'}^{(x)}||^2 - ||\mathbf{x}_k - \mathbf{m}_i^{(x)}||^2}{D}, 1 \leq i, i' \leq r,$$

where $D = \max_{k,k'} ||\mathbf{x}_k - \mathbf{x}_{k'}||^2, 1 \leq k, k' \leq N$ is the pointset diameter.

The non-normalized cluster assignment probabilities are given by

$$\hat{v}_i^{\mathbf{x}_k} = \exp(\rho(\min_{i'} \gamma_{(i,i')}(\mathbf{x}_k)))$$

and the normalized probabilities are then given by

$$v_i^{(\mathbf{x}_k)} = \frac{\hat{v}_i^{\mathbf{x}_k}}{\sum_{i'} \hat{v}_{i'}^{\mathbf{x}_k}} \tag{1}$$

A well known approximation to $\min_{i'} \gamma_{(i,i')}(\mathbf{x}_k)$ is the Kreisselmeier-Steinhauser (KS) envelope function [1, 4, 11] given by

$$KS_i(\mathbf{x}_k) = \frac{-1}{\rho} \ln \Big[\sum_{i'=1}^{r} \exp(-\rho \gamma_{(i,i')}(\mathbf{x}_k)) \Big],$$

where $\rho \gg 0$. The KS function is a smooth function that is infinitely differentiable. Using this the cluster membership indicators are redefined as:

$$v_i^{(\mathbf{x}_k)} = \frac{\exp\Big[\rho KS_i(\mathbf{x}_k)\Big]}{\sum_{i'=1}^{r} \exp\Big[\rho KS_{i'}(\mathbf{x}_k)\Big]} = \frac{\exp(-\frac{\rho}{D}||x_k - m_i||^2)}{\sum_{i'=1}^{r} \exp(-\frac{\rho}{D}||x_k - m_i'||^2)} \tag{2}$$

The cluster memberships for the "right" window, $v_j^{(\mathbf{y}_k)}$, are also smoothed similarly.

Note that ρ/D is the width of the Gaussian kernel used for approximation, however the KS-function helps tease out how the width must be set in order to achieve a certain quality of approximation. Furthermore, D is completely determined by the data but ρ is a user-settable parameter, and precisely what we can tune. The KS-function provides bounds on the error with which it approximates $\min_{i'} \gamma_{(i,i')}(\mathbf{x}_k)$, given as follows

$$\min_{i'} \gamma_{(i,i')}(\mathbf{x}_k) - \frac{\ln(N)}{\rho} \leq KS_i(\mathbf{x}_k) \leq \min_{i'} \gamma_{(i,i')}(\mathbf{x}_k)$$

The above inequality shows that the value of ρ can be determined for a given error precision. Furthermore, we can use any other rapidly decaying function to assign the probabilities (i.e. in place of the exponential) and the KS-function would again help us smooth the resulting assignments, but will yield assignment probabilities that are quite different from the traditional Gaussian kernel. In this sense, the KS-approximation is a versatile approach to smooth a variety of functions.

1.3 Objective Function and Regularization

As discussed in the main text, the objective function can be written as:

$$\begin{aligned}\mathcal{F} &= -\frac{1}{r}\sum_{i=1}^{r} H(R_i) - \frac{1}{c}\sum_{j=1}^{c} H(C_j), \\ &= -\frac{1}{r}\sum_{i=1}^{r} H(\beta|\alpha = i) - \frac{1}{c}\sum_{j=1}^{c} H(\alpha|\beta = j)\end{aligned}$$

where the entropy terms derive from KL-divergences w.r.t. the uniform distribution. Minimizing the function $\mathcal{F}$ should ideally yield clusters that are independent across windows and local within each window. However, using smooth cluster prototypes gives rise to an alternative minimum solution where each data sample is assigned with uniform probability to each cluster. For example, recall the 3×3 uniform contingency table example, where each of 18 samples can be assigned to 3 row clusters and 3 column clusters with probability $[1/3, 1/3, 1/3]$ and the estimate of the count matrix from these soft counts would still be uniform in each cell ($\sum_k v_i^{(\mathbf{x}_k)} v_j^{(\mathbf{y}_k)} = 2$). To avoid degenerate solutions such as these, we require maximum deviation of individual data vector probabilities ($v_i^{(\mathbf{x}_k)}$ and $v_j^{(\mathbf{y}_k)}$) from the uniform distribution over the number of clusters. This leads to the regularized objective function:

$$\begin{aligned}\mathcal{F} \quad = \quad & \frac{\lambda}{r}\sum_{i=1}^{r} D_{KL}\left(p_{R_i}||U(\tfrac{1}{c})\right) + \frac{\lambda}{c}\sum_{j=1}^{c} D_{KL}\left(p_{C_j}||U(\tfrac{1}{r})\right) \\ & -\frac{1}{N}\sum_{k=1}^{N} D_{KL}\left(p_{V^{(\mathbf{x}_k)}}||U(\tfrac{1}{r})\right) - \frac{1}{N}\sum_{k=1}^{N} D_{KL}\left(p_{V^{(\mathbf{y}_k)}}||U(\tfrac{1}{c})\right), \end{aligned} \tag{3}$$

where λ is the weight, set to a value greater than 1, to give more emphasis to minimizing the row and column distributions. This also enforces equal cluster sizes.

The role of λ is to enforce a 'balancing constraint' on the clusters (i.e., approximately equal cluster sizes) and to prevent samples from being assigned to multiple clusters. Other approaches have focused on explicitly capturing these aspects by an objective function but here we intend λ to be a regularization parameter, intended to avoid degenerate solutions. Hence the exact value of λ is not as crucial as the regime in which we conduct the optimization. In order to adjust λ, we vary its value over a range (typically [1,2] in small step sizes). Based on experimentation, the cluster assignments of most gene vectors (more than 90%) do not change after a particular value of λ and we use this criterion to set λ. All the terms in the definition of $\mathcal{F}$ above can be calculated in terms of the smoothed cluster membership probabilities $v_i^{(\mathbf{x}_k)}$ and $v_j^{(\mathbf{y}_k)}$, which are in turn calculated in terms of the cluster prototypes $\mathbf{m}_i^{(x)}$ and $\mathbf{m}_j^{(y)}$. Thus the objective function $\mathcal{F}$ is effectively parametrized in terms of the cluster prototypes, and the problem of finding independent clusters now reduces to finding the cluster prototypes that optimize the objective function.

1.4 Optimization

The gradient of $\mathcal{F}$ with respect to the prototypes $\mathbf{m}_i^{(x)}$ is given by

$$\begin{aligned}\nabla_{\mathbf{m}_i^{(x)}}\mathcal{F} = \frac{1}{\ln(2)}\sum_{i'=1}^{r}\sum_{k=1}^{N}\Bigg(\frac{\lambda}{r}\Bigg\{\sum_{j=1}^{c}\Big[1+\ln\Big(\frac{\sum_{k'=1}^{N} v_{i'}^{(\mathbf{x}_{k'})} v_j^{(\mathbf{y}_{k'})}}{\sum_{k'=1}^{N} v_{i'}^{(\mathbf{x}_{k'})}}\Big/\frac{1}{c}\Big)\Big]\cdot \\ \Big[\frac{v_j^{(\mathbf{y}_k)}}{\sum_{k'=1}^{N} v_{i'}^{(\mathbf{x}_{k'})}} - \frac{\sum_{k'=1}^{N} v_{i'}^{(\mathbf{x}_{k'})} v_j^{(\mathbf{y}_{k'})}}{\sum_{k'=1}^{N}(v_{i'}^{(\mathbf{x}_{k'})})^2}\Big]\Bigg\} \\ -\frac{\lambda}{c}\Bigg\{\sum_{j=1}^{c}\Big[1+\ln\Big(\frac{\sum_{k'=1}^{N} v_{i'}^{(\mathbf{x}_{k'})} v_j^{(\mathbf{y}_{k'})}}{\sum_{k'=1}^{N} v_j^{(\mathbf{y}_{k'})}}\Big/\frac{1}{r}\Big)\Big]\Big[\frac{v_j^{(\mathbf{y}_k)}}{\sum_{k'=1}^{N} v_j^{(\mathbf{y}_{k'})}}\Big]\Bigg\} \\ -\frac{1}{N}\Big[1+\ln\Big(v_{i'}^{(\mathbf{x}_k)}\Big/\frac{1}{r}\Big)\Big]\Bigg)\nabla_{\mathbf{m}_i^{(x)}} v_{i'}^{(\mathbf{x}_k)},\end{aligned} \tag{4}$$

input $\mathcal{T} = (t_1, t_2, \ldots, t_l)$: Given time series data sequence.
input l_{min}: Minimum window length.
input l_{max}: Maximum window length.

Step 1: Define the set of windows starting from time point t_a, $S_{t_a} = \{w_{t_a}^{t_b} | l_{min} \leq t_b - t_a + 1 \leq l_{max}\}$.
Step 2: Construct a directed acyclic graph where each $w_{t_a}^{t_b}$ is a node and a directed edge exists from $w_{t_a}^{t_b}$ to the windows $w_{t_{b+1}}^{t_c} \in S_{t_{b+1}}$.
Step 3: Cluster the adjacent windows $w_{t_a}^{t_b}$ and $w_{t_{b+1}}^{t_c}$ by minimizing the objective function in Eq.(3). Let $\mathcal{F}_{w_{t_a}^{t_b}, w_{t_{b+1}}^{t_c}}$ be the final value of the objective function. Assign $\mathcal{F}_{w_{t_a}^{t_b}, w_{t_{b+1}}^{t_c}}$ as the edge weight between the nodes represented by $w_{t_a}^{t_b}$ and $w_{t_{b+1}}^{t_c}$.
Step 4: Let $E_{t_l} = \{w_{t_k}^{t_l} | l_{min} \leq t_l - t_k + 1 \leq l_{max}\}$ be the set of windows ending in the last time point t_l. For each window starting at the first time point, $w_{t_1}^{t_b} \in S_{t_1}$, calculate the minimum cost path to all $w_{t_k}^{t_l} \in E_{t_l}$.
Step 5: Calculate $D_{Seg} = \mathcal{F}_{\{w_{t_1}^{t_a}, w_{t_{a+1}}^{t_b}\}} + \mathcal{F}_{\{w_{t_{b+1}}^{t_c}, w_{t_{c+1}}^{t_d}\}} + \ldots + \mathcal{F}_{\{w_{t_j}^{t_k}, w_{t_{k+1}}^{t_l}\}}$ for each shortest path in step 4.
Step 6: Return the path with minimum D_{Seg} .

Figure 1: Algorithm for segmenting a time series

where

$$\nabla_{\mathbf{m}_i^{(x)}} v_i^{(\mathbf{x}_k)} = \frac{2\rho(\mathbf{x}_k - \mathbf{m}_i^{(x)})}{D} v_i^{(\mathbf{x}_k)} (\delta_{i',i} + v_i^{(\mathbf{x}_k)}) \tag{5}$$

Here $\delta_{i',i}$ is the Kronecker delta. The index variables i, i', and i'' are over the clusters in the $\mathbf{x}$ vectors, j over the clusters in the $\mathbf{y}$ vectors, and k and k' over the data vectors. The gradients with respect to the prototypes $\mathbf{m}_j^{(y)}$ are calculated analogously.

Optimization of $\mathcal{F}$ is performed using the augmented Lagrangian algorithm with simple bound constraints on the cluster prototypes using the FORTRAN package LANCELOT [2]. The initial cluster prototypes are set using individual k-means clusters in each window. The augmented Lagrangian algorithm iteratively improves these initial prototypes till a local minimum of the objective function is attained.

1.5 Dynamic programming

We can now present the algorithm for tiling a given time course into a series of windows such that neighboring windows capture breakpoints of significant re-organization (see Fig. 1). Here, l_{min} and l_{max} are the minimum and maximum window lengths. We create all possible 'tiles' of the entire time course that satisfy these constraints and then evaluate every consecutive pair of them. The evaluation consists of applying our clustering framework and determining the minimized value of $\mathcal{F}$. These objective functions are then summed over an entire segmentation and used to evaluate one segmentation over another. Computationally, this reduces to a shortest path algorithm where each edge length is given by the minimized value of $\mathcal{F}$. Each optimization can be performed in a matter of a few seconds on a desktop computer so that the entire segmentation is computable in a few minutes.

2 Evaluation and Assessment

We evaluate our clusterings and segmentations in five ways: cluster stability, cluster reproducibility, functional enrichment, segmentation quality, and segmentation sensitivity. Cluster stability and cluster reproducibility are used to filter the patterns obtained by the segmentation algorithm whereas the other three criteria are used to evaluate the quality of segmentation.

We assess **cluster stability** using a bootstrap procedure to determine significance of genes brought together. Recall that each window except the first and last windows has two sets of clusters, one set independent with respect to the previous window and the other independent with respect to the next window. We are interested in the genes that are significantly clustered together in these two sets of clusters, as they represent the genes that are specific to the window under consideration. We calculate a contingency table between these two clusterings for each window (excluding the first and the last window) in which each cell represents the number of genes that are together across the two independent sets of clusters. We randomly sample 1000 pairs of clusterings within each window (with cluster sizes same as the two independent clusterings) and compute their contingency tables. By the central limit theorem, the distribution of counts in each cell of the table is approximately normal (also verified using a Shapiro-Wilk normality test with $p = 0.05$). We now evaluate each cell of the actual contingency table with respect to the corresponding random distribution and retain only those cells that have more genes than that observed at random with $p < 0.05$ (Bonferroni corrected with the number of cross clusters to account for multiple hypothesis testing). To ensure **reproducibility** of clusters, we retain only those genes in each significant cell of the contingency table that are together in more than 150 of the 200 clusterings (conducted with different initializations). For the first and last windows, which have only 100 randomly initialized clusterings, we retain those genes that are clustered together in more than 75 of the 100 clusterings. At this stage, for each segment we obtain a contingency table with significant cells representing the group of genes specific to the particular segment, which we can refer to as cross-cluster. We perform **functional enrichment** using the GO biological process ontology (since we are tracking biological processes) over each of these cross clusters. A hypergeometric p-value is calculated for each GO biological process term, and an appropriate cutoff is chosen using a false discovery rate (FDR) q-level of 0.01.

The **segmentation quality** is calculated as a partition distance [6] between the "true" segmentation (from the literature of the YMC and YCC) to the segmentations computed by our algorithm. We view each window as a set of time points so that a segmentation is a partition of time points. Given two segmentations S_1 and S_2, whose windows are indexed by the variables $w_{t_a}^{t_b}$ and $z_{t_c}^{t_d}$ respectively, the partition distance is given by:

$$\begin{aligned} P_D &= -\sum_{w_{t_a}^{t_b} \in S_1} \sum_{z_{t_c}^{t_d} \in S_2} |w_{t_a}^{t_b} \cap z_{t_c}^{t_d}| \log_2 \frac{|w_{t_a}^{t_b} \cap z_{t_c}^{t_d}|}{|w_{t_a}^{t_b}|} \\ &\quad -\sum_{z_{t_c}^{t_d} \in S_2} \sum_{w_{t_a}^{t_b} \in S_1} |w_{t_a}^{t_b} \cap z_{t_c}^{t_d}| \log_2 \frac{|w_{t_a}^{t_b} \cap z_{t_c}^{t_d}|}{|z_{t_c}^{t_d}|}. \end{aligned} \tag{6}$$

The **segmentation sensitivity** to variations in the number of clusters is calculated as the average of the ratios of KL-divergences between the segments to the maximum possible KL divergence between those segments. This latter figure is easy to compute as a function of the number of clusters, which is considered uniform throughout the segmentation. Suppose we have $|S|$ windows in a given segmentation $S = \{w_{t_1}^{t_a}, w_{t_{a+1}}^{t_b}, \ldots, w_{t_{j+1}}^{t_k}, w_{t_{k+1}}^{t_l}\}$ with c clusters in each window. Let $\mathcal{F}_{max}$ be the objective function

value for the maximally similar clustering (the $c \times c$ diagonal contingency table). Then the measure we compute is

$$D_{avg} = \frac{1}{|S|-1}\left[\frac{\mathcal{F}_{\{w_{t_1}^{t_a}, w_{t_{a+1}}^{t_b}\}}}{\mathcal{F}_{max}} + \frac{\mathcal{F}_{\{w_{t_{b+1}}^{t_c}, w_{t_{c+1}}^{t_d}\}}}{\mathcal{F}_{max}} + \ldots + \frac{\mathcal{F}_{\{w_{t_j}^{t_k}, w_{t_{k+1}}^{t_l}\}}}{\mathcal{F}_{max}}\right], \tag{7}$$

where $\mathcal{F}_{\{w_{t_a}^{t_b}, w_{t_{b+1}}^{t_c}\}}$ is the optimal objective function value obtained by clustering the pair of adjacent windows $w_{t_a}^{t_b},\ w_{t_{b+1}}^{t_c}$. Observe that this criterion is different from the actual criterion optimized during the segmentation. D_{avg} compares our segmentation (which identifies dissimilar sets of clusters) to the case when there are exactly similar clusters. Lower values of this ratio indicate that the segmentation captures maximal independence between adjacent segments while higher values indicate the clusters obtained are more similar in adjacent segments.

3 Datasets and pre-processing

Our datasets came from a variety of sources. For each dataset described below, we retained only genes that have an annotation in the GO (Gene Ontology) biological process taxonomy (revision 4.205 of GO released on 14 March 2007), log transformed (base 10) their expression values and normalized them such that the mean expression of each gene across all time points is zero. The YMC1 dataset [10] consists of 36 time points collected over 3 continuous cycles. The original dataset consists of 6555 unique genes from the *S. cerevisiae* genome from which after preprocessing as described above we retained 3602 genes. We also analyzed another yeast metabolic cycle (YMC2) dataset [3] with 32 time points collected over 3 continuous cycles. (While in YMC1 [10] the authors claim that the cycle length is approximately 5 hours, in YMC2 [3], the authors claim that this cycle length is approximately 40 minutes.) Here we again retained 3602 genes. As our third dataset, we analyzed the well known yeast cell cycle dataset from experiments performed bySpellman et al. [8]. There are three components to the Spellman yeast cell cycle (YCC) data, following three different cell synchronization treatments with α-factor, cdc 15 and elutriation. We describe our analysis of the α-factor dataset that has 6076 genes with 18 time points over approximately 2 cycles. Our preprocessing results in a universal set of 2196 genes. Finally, we analyzed datasets from the experiments conducted by Shapira et al. [7], who studied the effects of oxidative stress induced by hydrogen peroxide (HP) and Menadione (MD) on the yeast cell cycle. We analyzed the datasets where HP and MD were added to the cells at 25 minutes after release from G1 arrest. The cells treated with HP were arrested in the subsequent G2/M phase while those treated with MD go through one cell cycle and were arrested in the G1 phase of next cycle. The HP dataset has 20 time points while the MD dataset has 14 time points. After preprocessing, we obtained a final set of 2471 genes in HP, and 2247 genes in MD datasets.

The above datasets are segmented using our dynamic programming algorithm. We used different settings for the numbers of clusters and different thresholds for minimum and maximum possible window lengths to search in the space of possible segmentations. Besides the number of clusters in each segment, and minimum/maximum constraints on window lengths, weparameterized the segmentation algorithm with a parameter λ that controls the sizes of the clusters in the resulting segmentations and can be adjusted to yield approximately equal cluster sizes. For YMC1, we experimented with the number of clusters in each segment ranging from 3 to 15, a minimum window length of 4 and max imum window length of 7, and $\lambda = 1.4$. For YMC2, the number of clusters is varied between 3 and 15, minimum and maximum window lengths used are 3 and 6 respectively, a nd $\lambda = 1.35$. For the YCC, we ranged from 3 to 15 clusters in each window

with a minimum window length of 3 and maximum window length of 5, and $\lambda = 1.25$. For segmenting the HP and MD datasets, the number of clusters is varied between 3 and 15, minimum and maximum window lengths used are 3 and 7 respectively, and $\lambda = 1.55$. After the segmentation reveals windows and clusters of genes in each window, we perform functional enrichment over the selected sets of genes. A hypergeometric p-value is calculated for each GO biological process term, and an appropriate cutoff is chosen using false discovery rate (FDR) q-level of 0.01 [9].

4 Comparisons with Biclustering

It is instructive to compare our segmentation and temporal process building approach with classical biclustering algorithms. Biclustering algorithms, even those that obey the temporal order of time points, will find groups of genes and groups of time points but will not yield a representation of the entire time course. A segmentation, on the other hand, is a partition of the entire time course. Consider for instance applying a biclustering algorithm such as e-CCC [5] and our segmentation algorithm to the yeast cell cycle data (see Figure 3 of our paper: time points 1–9). The classical metaphor for biologists to comprehend cell cycle data is to view it in terms of phases – something our algorithm naturally reveals. e-CCC, on the other hand, merely brings out a laundry list of biclusters which have to be processed to understand phases and transitions. But this is not possible and there are serious shortcomings:

Lack of a complete partition of the time course: e-CCC when run on the yeast cell cycle data would return results such as:

DNA replication initiation genes	3–5
strand elongation genes	3–7
metaphase activated genes	6–8

Note that because such an algorithm only looks for biclusters, the time points can overlap. Furthermore, the genes corresponding to these biclusters could overlap as well. Finally, and more importantly, some time points might not be represented in the answer (such as time points 1, 2, and 9, in the above case). These algorithms hence make no attempt to present an integrated picture of the temporal patterns manifest in the data, such as to find transition points in a global interval. Our segmentation algorithm goes considerably further. It posits that there are key time point boundaries around which genes re-group and re-organize. In addition, within a segment, we do not assume that there is a single process/function that is manifest but rather a group (set) of processes. It is these groups that get re-defined around the time point boundaries. For the above data, we get our result as:

cytokinesis	1–3
DNA replication	1–3
strand elongation	4–6
G1/S-specific transcription	4–6
mitotic spindle elongation	7–9
metaphase/anaphase transition	7–9

Note that all time points are represented in the answer but each time point participates in only one segment. Within each segment is enriched a set of processes. Using an entire segmentation such as this, we obtain an integrated view of the key phases underlying the dataset.

Inability to support temporal model building and inference: A more serious concern is that biclusters from algorithms such as e-CCC cannot be further processed into a complete temporal model, such as a finite state machine, Kripke structure, or other automata. This is a key contribution of our paper. Our segmentation and enrichment of each phase of the discovered segments immediately results in a Kripke structure (see Figure 1 of our paper). More importantly, we are able to combine such temporal models across experiments into an integrated process model (see Figure 2 of our paper). In summary, to use an oft-quoted metaphor, the biclustering algorithm gives us pieces of the puzzle which might not fit together and where some pieces are missing. The segmentation algorithm gives us pieces that fit together and which when composed yield the entire picture.

5 Gantt charts

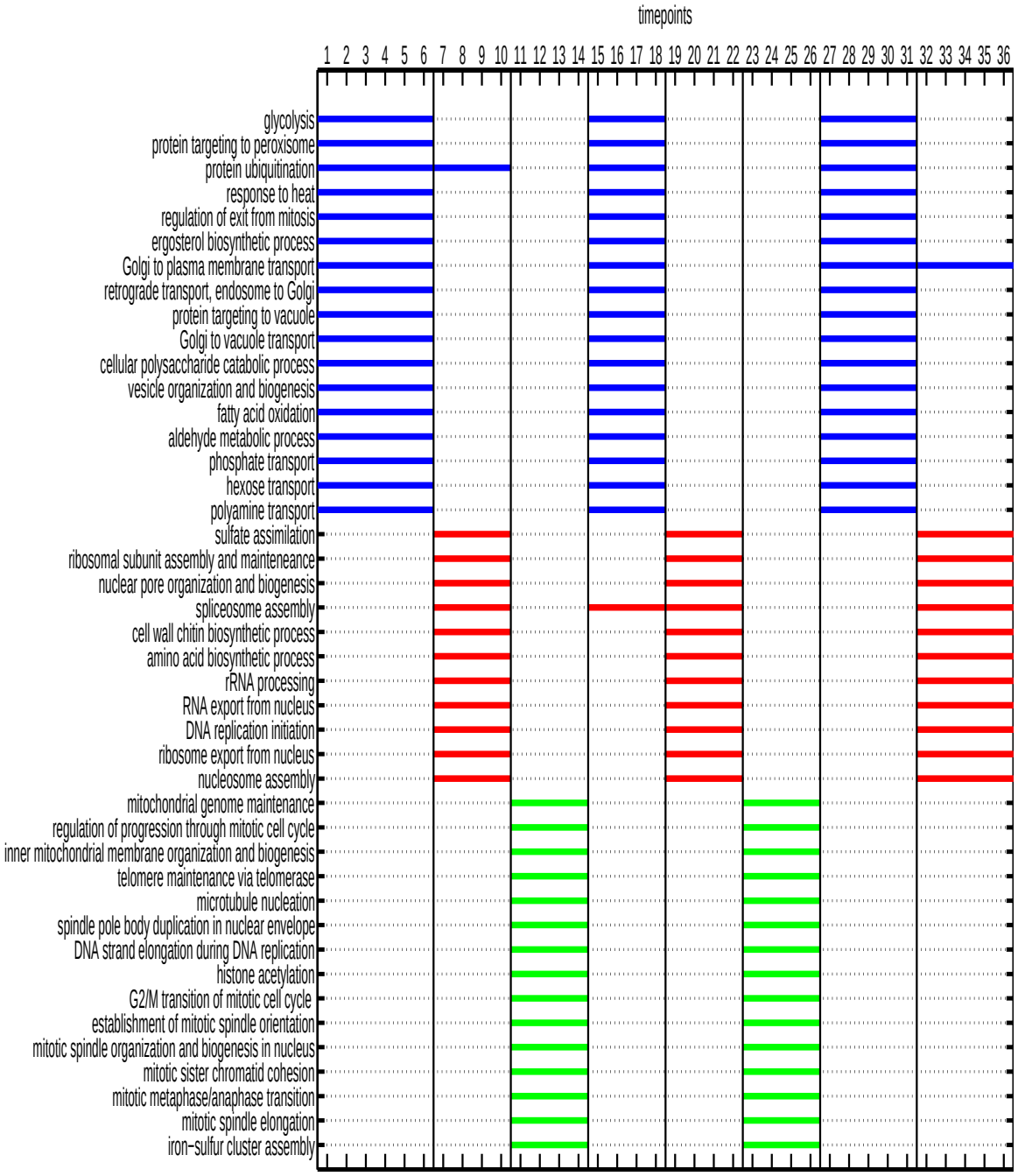

Figure 2: Gantt chart resulting from segmentation of the YMC1 dataset.

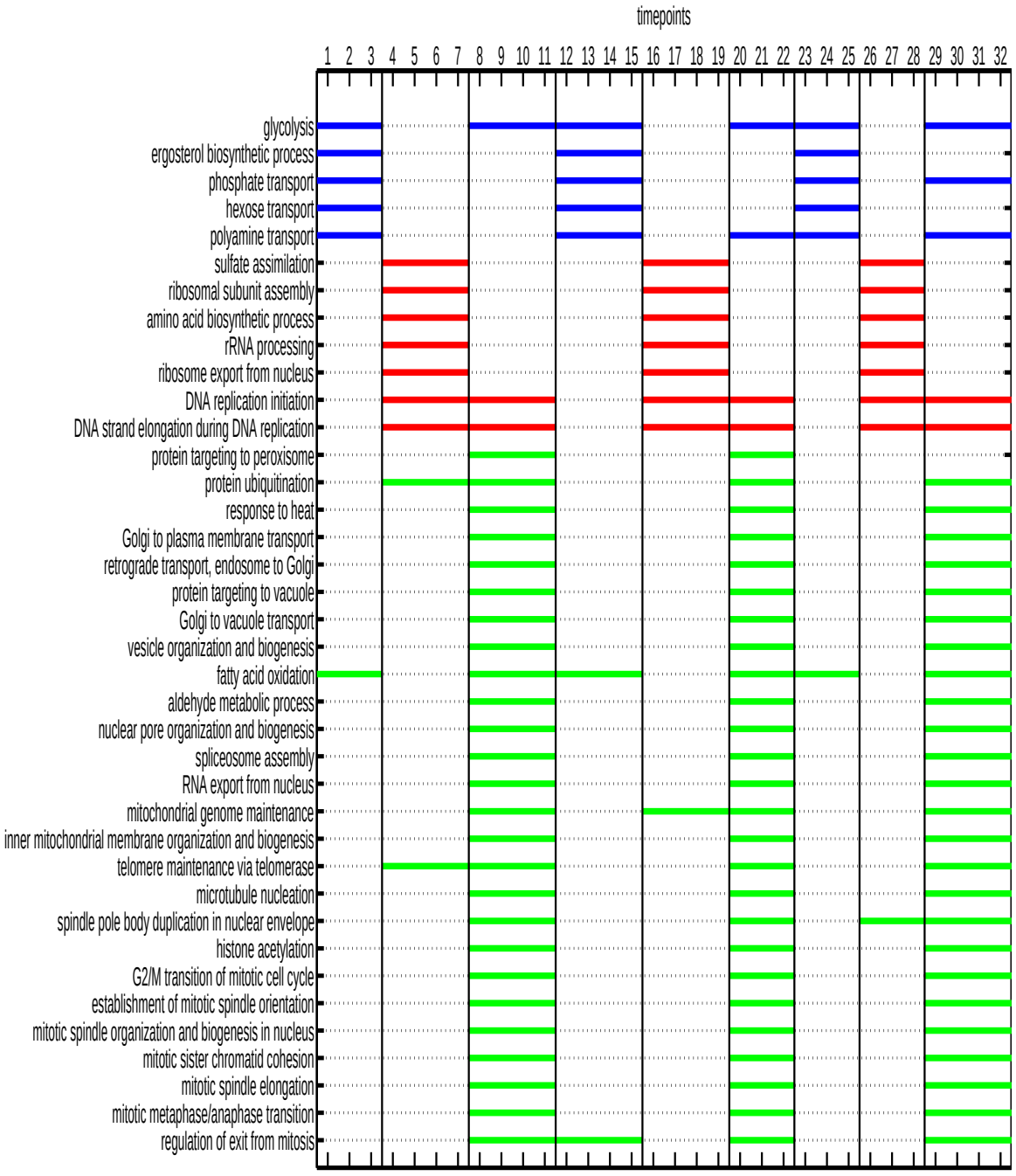

Figure 3: Gantt chart resulting from segmentation of the YMC2 dataset. Compare with Fig. 2.

References

[1] J. F. M. Barthelemy and M. F. Riley. Improved multi-level optimization approach for the design of complex engineering systems. *AIAA J.*, 26:353–360, 1988.

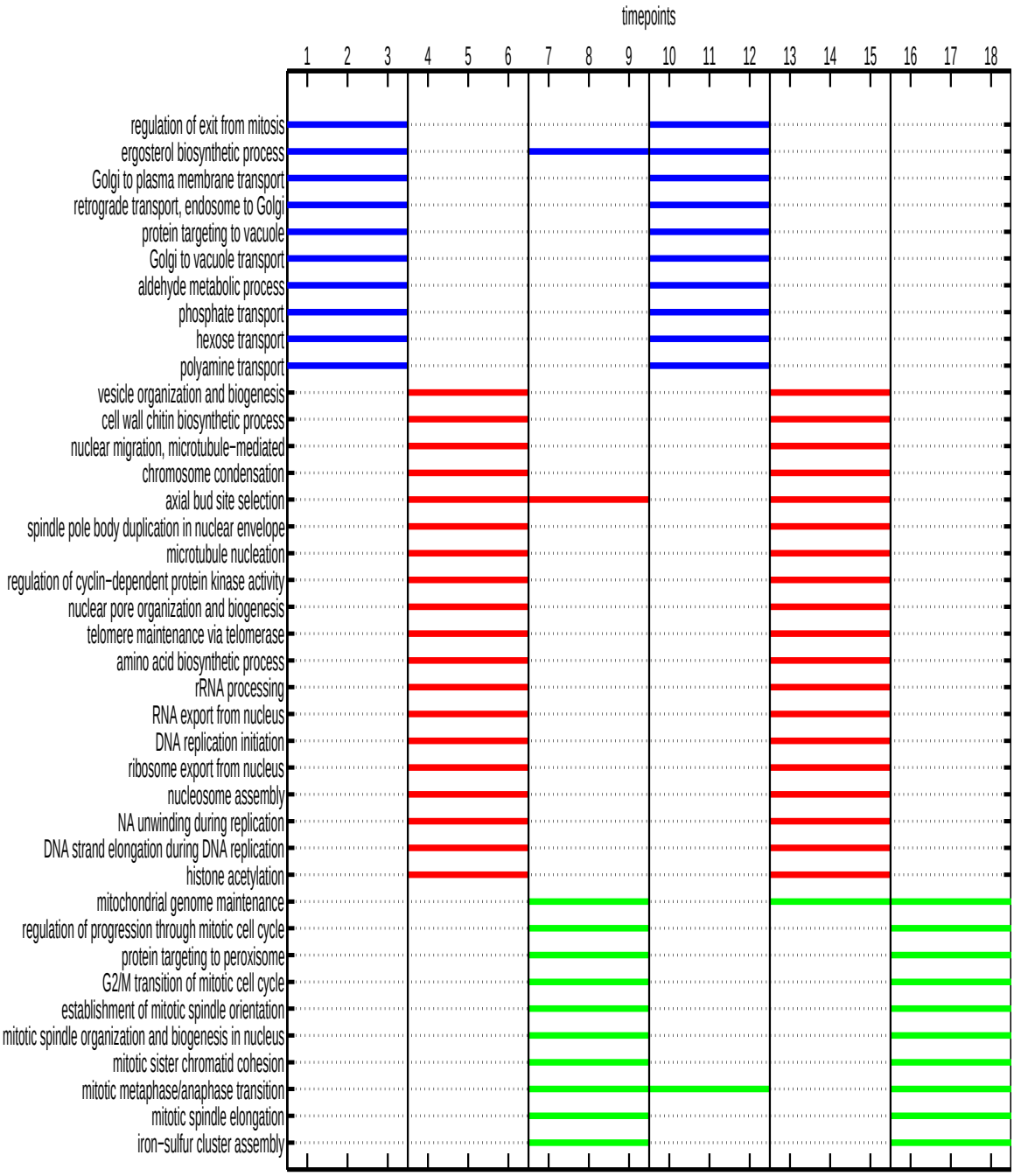

Figure 4: Gantt chart resulting from segmentation of the YCC dataset.

[2] A.R. Conn, N.I.M. Gould, and P.L. Toint. *LANCELOT: A Fortran Package for Large-scale Nonlinear Optimization (Release A)*, volume 17. Springer Verlag, 1992.

[3] R.R. Klevecz, J. Bolen, G. Forrest, and D. B. Murray. A genomewide oscillation in transcription gates dna replication and cell cycle. *PNAS*, 101(5):1200–1205, 2004.

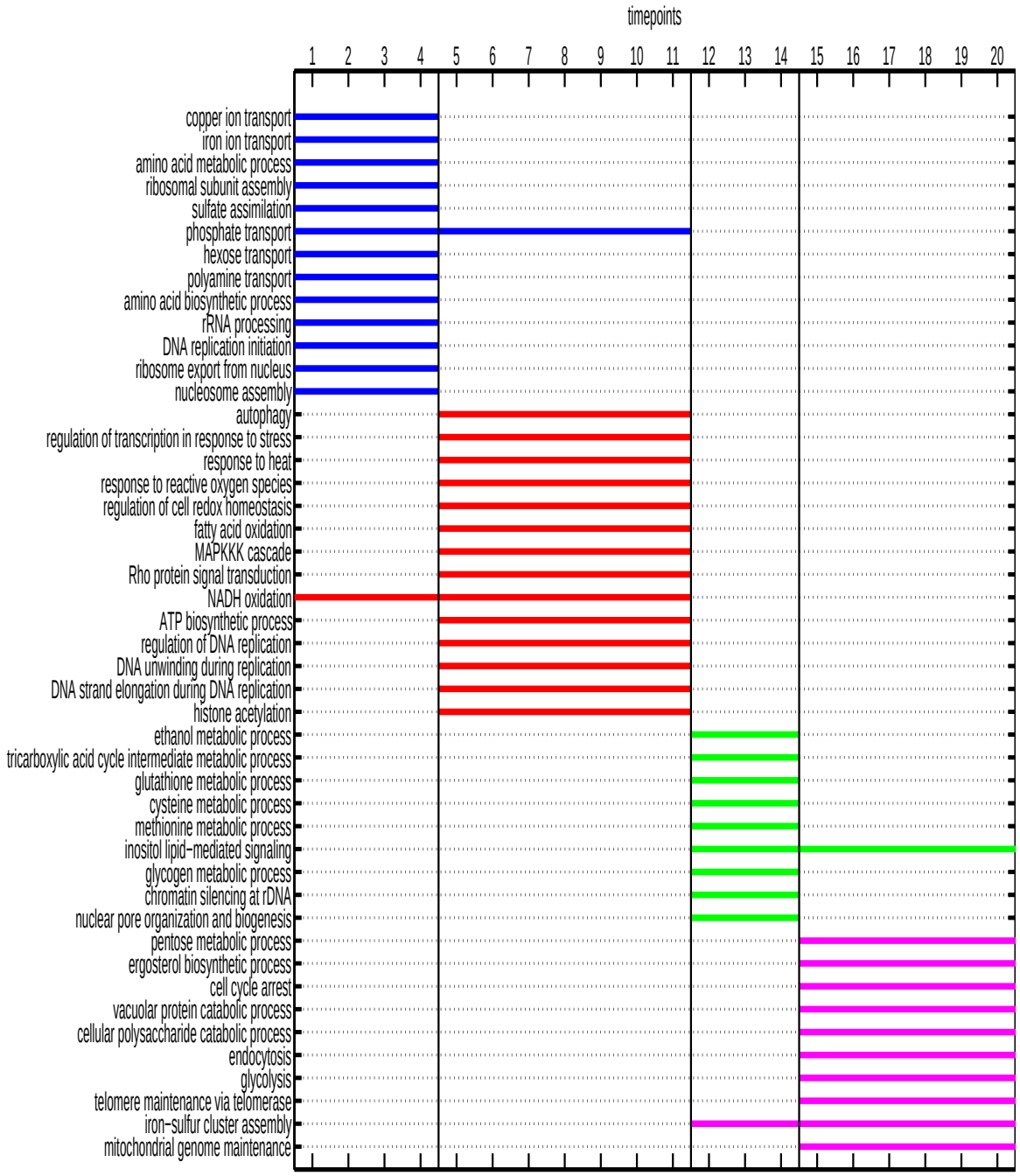

Figure 5: Gantt chart resulting from segmentation of the HP dataset.

[4] G. Kreisselmeier and R. Steinhauser. Systematic control design by optimizing a vector performance index. In *IFAC Symp. on computer aided design of control systems*, pages 113–117, 1979.

[5] S.C. Madeira and A.L. Oliveira. A Polynomial Time Biclustering Algorithm for Finding Approximate Expression Patterns in Gene Expression Time Series. *Algorithms for Molecular Biology*, Vol. 4, 2009.

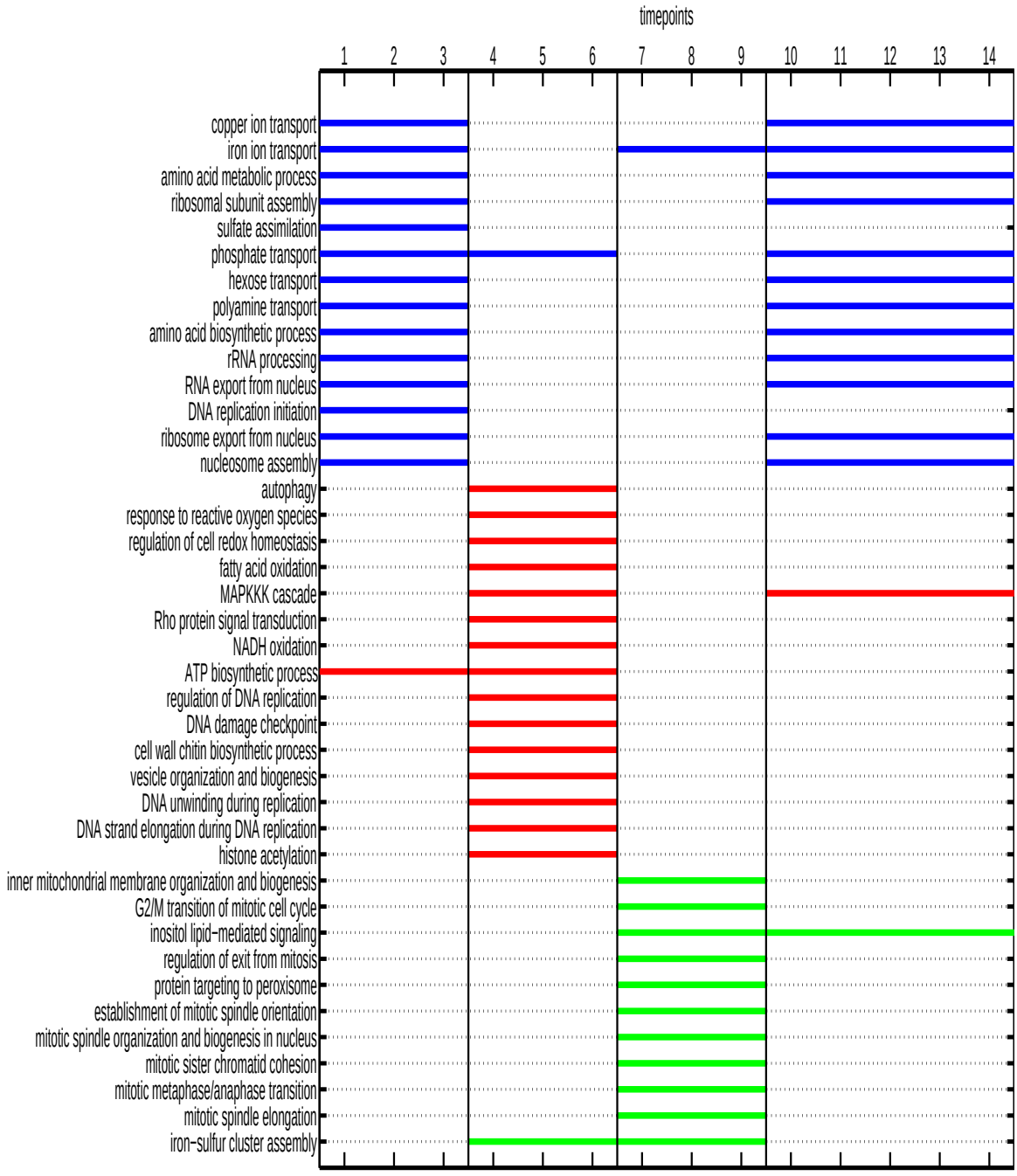

Figure 6: Gantt chart resulting from segmentation of the MD dataset.

[6] R.L. De Mántaras. A Distance-Based Attribute Selection Measure for Decision Tree Induction. *Machine Learning*, 6(1):81–92, 1991.

[7] M. Shapira, E. Segal, and D. Botstein. Disruption of Yeast Forkhead-associated Cell Cycle Transcription by Oxidative Stress. *Molecular Biology of the Cell*, 15(12):5659–5669, 2004.

[8] P.T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B. Eisen, P.O. Brown, D. Botstein, and B. Futcher. Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast Saccharomyces cerevisiae by Microarray Hybridization. *Molecular Biology of the Cell*, 9(12):3273–3297, 1998.

[9] J.D. Storey and R. Tibshirani. Statistical significance for genomewide studies. *PNAS*, 100(16):9440–9445, 2003.

[10] B.P. Tu, A. Kudlicki, M. Rowicka, and S.L. McKnight. Logic of the Yeast Metabolic Cycle: Temporal Compartmentalization of Cellular Processes. *Science*, 310(5751):1152–1158, 2005.

[11] L. T. Watson and R. T. Haftka. Modern homotopy methods in optimization. *Computer methods in appl. mechanics and engg.*, 74:289–305, 1989.