Edinburgh Research Explorer

Learning differential module networks across multiple experimental conditions

Citation for published version:

Link:
Link to publication record in Edinburgh Research Explorer

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 08. Mar. 2019
Learning differential module networks across multiple experimental conditions

Pau Erola, Eric Bonnet and Tom Michoel

Abstract Module network inference is a statistical method to reconstruct gene regulatory networks, which uses probabilistic graphical models to learn modules of coregulated genes and their upstream regulatory programs from genome-wide gene expression and other omics data. Here we review the basic theory of module network inference, present protocols for common gene regulatory network reconstruction scenarios based on the Lemon-Tree software, and show, using human gene expression data, how the software can also be applied to learn differential module networks across multiple experimental conditions.

Key words: gene regulatory network inference, module networks, differential networks, Bayesian analysis

1 Introduction

Complex systems composed of a large number of interacting components often display a high level of modularity, where independently functioning units can be observed at multiple organizational scales [1]. In biology, a module is viewed as a discrete entity composed of many types of molecules and whose function is sepa-
rable from that of other modules [2]. The principle of modularity plays an essential role in understanding the structure, function and evolution of gene regulatory, metabolic, signaling and protein interaction networks [3]. It is therefore not surprising that functional modules also manifest themselves in genome-wide data. Indeed, from the very first studies examining genome-wide gene expression levels in yeast, it has been evident that clusters of coexpressed genes, i.e. sharing the same expression profile over time or across different experimental perturbations, reveal important information about the underlying biological processes [4, 5]. Module network inference takes this principle one step further, and aims to infer simultaneously co-expression modules and their upstream regulators [6, 7]. From a statistical perspective, modularity allows to reduce the number of model parameters that need to be determined, because it is assumed that genes belonging to the same module share the same regulatory program, and therefore allows to learn more complex models, in particular non-linear probabilistic graphical models [8], than would otherwise be possible.

While module networks were originally introduced to infer gene regulatory networks from gene expression data alone [6], the method has meanwhile been extended to also include expression quantitative trait loci data [9, 10], regulatory prior data [11], microRNA expression data [12], clinical data [13], copy number variation data [14, 15] or protein interaction networks [16]. Furthermore, the method can be combined with gene-based network inference methods [17, 18]. Finally, the module network method has been applied in numerous biological, biotechnological and biomedical studies [19–29].

An area of interest that has received comparatively limited attention to date concerns the inference of differential module networks. Differential networks extend the concept of differential expression, and are used to model how coexpression, regulatory or protein-protein interaction networks differ between two or more experimental conditions, cell or tissue types, or disease states [30, 31]. Existing differential network inference methods are mainly based on pairwise approaches, either by testing for significant differences between correlation values in different conditions, or by estimating a joint graph from multiple data sets simultaneously using penalized likelihood approaches [32–35]. The inference of differential module networks is more challenging, because it requires a matching or comparable set of modules across the conditions of interest. A related problem has been addressed in a study of the evolutionary history of transcriptional modules in a complex phylogeny, using an algorithm that maps modules across species and allows to compare their gene assignments [36].

In this chapter, we review the theoretical principles behind module network inference, explain practical protocols for learning module networks using the Lemon-Tree software [15], and show in a concrete application on human gene expression data how the software can also be used to infer differential module networks using a similar principle as in [36].
2 Module network inference: theory and algorithms

2.1 The module network model

Module networks are probabilistic graphical models [7, 8] where each gene $g_i$, $i \in \{1, \ldots, G\}$, is represented by a random variable $X_i$ taking continuous values. In a standard probabilistic graphical model or Bayesian network, it is assumed that the distribution of $X_i$ depends on the expression level of a set of regulators $P_i$ (the “parents” of gene $i$). If the causal graph formed by drawing directed edges from parents to their targets is acyclic, then the joint probability distribution for the expression levels of all genes can be written as a product of conditional distributions,

$$p(x_1, \ldots, x_G) = \prod_{i=1}^{G} p(x_i | \{x_j: j \in P_i\}). \quad (1)$$

In data integration problems, we are often interested in explaining patterns in one data type (e.g. gene expression) by regulatory variables in another data type (e.g. transcription factor binding sites, single nucleotide or copy number variations, etc.). In this case, the causal graph is bipartite, and the acyclicity constraint is satisfied automatically.

In a module network, we assume that genes are partitioned into modules, such that genes in the same module share the same parameters in the distribution function (1). Hence a module network is defined by a partition of $\{1, \ldots, G\}$ into $K$ modules $\mathcal{M}_k$, a collection of parent genes $P_k$ for each module $k$, and a joint probability distribution

$$p(x_1, \ldots, x_G) = \prod_{k=1}^{K} \prod_{i \in \mathcal{M}_k} p(x_i | \{x_j: j \in P_k\}). \quad (2)$$

In a module network, only one conditional distribution needs to be parameterized per module, and hence it is clear that if $K \ll G$, the number of model parameters in eq. (2) is much smaller than in eq. (1). Moreover, data from genes belonging to the same module are effectively pooled, leading to more robust estimates of these model parameters. This is the main benefit of the module network model.

In principle, any type of conditional distribution can be used in eq. (2). For instance, in a linear Gaussian framework [8], one would assume that each gene is normally distributed around a linear combination of the parent expression levels. However, the pooling of genes into modules allows for more complex, non-linear models to be fitted. Hence it was proposed that the conditional distribution of the expression level of the genes in module $k$ is normal with mean and standard deviation depending on the expression values of the parents of the module through a regression tree (the “regulatory program” of the module) [6] (Figure 1). The tests on the internal nodes of the regression tree are usually defined to be of the form $x \geq v$ or not, for a split value $v$, where $x$ is the expression value of the parent associated to the node.
Fig. 1 Example of a module and regulatory decision tree inferred from yeast data, with Hap4 assigned as a top regulator. Genes known to be regulated by Hap4 in YEASTRACT are marked in blue and those involved in respiration are marked in orange. Reused from Joshi et al., Module networks revisited: computational assessment and prioritization of model predictions, Bioinformatics, 2009, 25(4):490–496 [37], by permission of Oxford University Press.

Given a module network specification $M$, consisting of gene module assignments, regulatory decision trees, and normal distribution parameters at the leaf nodes, the probability density of observing an expression data matrix $X = (x_{im}) \in \mathbb{R}^{G \times N}$ for $G$ genes in $N$ samples is given by

$$P(X | M) = \prod_{m=1}^{N} \prod_{k=1}^{K} \prod_{i \in \mathcal{E}_k} p(x_{im} | \{x_{jm} : j \in \mathcal{P}_k\}) = \prod_{k=1}^{K} L_k \prod_{\ell=1}^{L_k} \prod_{i \in \mathcal{E}_\ell} p(x_{im} | \mu_\ell, \sigma_\ell),$$

where $L_k$ is the number of leaf nodes of module $k$’s regression tree, $\mathcal{E}_\ell$ denotes the experiments that end up at leaf $\ell$ after traversing the regression tree, and $(\mu_\ell, \sigma_\ell)$ are the normal distribution parameters at leaf $\ell$. The Bayesian model score is obtained by taking the log-marginal probability over the parameters of the normal distributions at the leaves of the regression trees with a normal-gamma prior:

$$S = \sum_k S_k = \sum_k \sum_\ell S_k(\mathcal{E}_\ell)$$

$$S_k(\mathcal{E}_\ell) = -\frac{1}{2} R_0^{(\ell)} \log(2\pi) + \frac{1}{2} \log(\frac{\lambda_0}{\lambda_0 + R_0^{(\ell)}}) - \log \Gamma(\alpha_0) + \log \Gamma(\alpha_0 + \frac{1}{2} R_0^{(\ell)}) + \alpha_0 \log \beta_0 - (\alpha_0 + \frac{1}{2} R_0^{(\ell)}) \log \beta_1$$

where $R_q^{(\ell)}$ are the sufficient statistics at leaf $\ell$. 
Learning differential module networks across multiple experimental conditions

\[ R_q^{\ell} = \sum_{m \in \ell \cap \mathcal{C}_k} \sum_i x_{i,m}^q, \quad q = 0, 1, 2, \]

and

\[ \beta_1 = \beta_0 + \frac{1}{2} \left[ R_2^{\ell} - \frac{(R_1^{\ell})^2}{R_0^{\ell}} \right] + \frac{\lambda_0 (R_1^{\ell} - \mu_0 R_0^{\ell})^2}{2 (\lambda_0 + R_0^{\ell}) R_0^{\ell}}. \]

Details of this calculation can be found in [38, 39].

### 2.2 Optimization algorithms

The first optimization strategy proposed to identify high-scoring module networks was a **greedy hill-climbing** algorithm [6]. This algorithm starts from an initial assignment of genes to coexpression clusters (e.g. using k-means), followed by assigning a new regulator to each module by iteratively finding the best (if any) new split of a current leaf node into two new leaf nodes given the current set of gene-to-module assignments, and reassigning genes between modules given the current regression tree, while preserving acyclicity throughout. The decomposition of the Bayesian score (eq. (3)) as a sum of leaf scores of the different modules allows for efficient updating after every regulator addition or gene reassignment.

An improvement to this algorithm was found, based on the observation that the Bayesian score depends only on the assignment of samples to leaf nodes, and not on the actual regulators or tree structure that induce this assignment [39]. Hence, a **decoupled greedy hill-climbing** algorithm was developed, where first the Bayesian score is optimized by two-way clustering of genes into modules and samples into leaves for each module, and then a regression tree is found for the converged set of modules by hierarchically merging the leave nodes and finding the best regulator to explain the split below the current merge. This algorithm achieved comparable score values as the original one, while being considerably faster [39].

Further analysis of the greedy two-way clustering algorithm revealed the existence of multiple local optima, in particular for moderate to large data sets (~1000 genes or more), where considerably different module assignments result in near-identical scores. To address this issue, a **Gibbs sampler** method was developed, based on the Chinese restaurant process [40], for sampling from the posterior distribution of two-way gene/sample clustering solutions [41]. By sampling an ensemble of multiple, equally probable solutions, and extracting a core set of ‘tight clusters’ (groups of genes which consistently cluster together), gene modules are identified that are more robust to fluctuations in the data and have higher functional enrichment compared to the greedy clustering strategies [37, 41].

Finally, the Gibbs sampling strategy for module identification was complemented with a probabilistic algorithm, based on a logistic regression of sample splits on candidate regulator expression levels, for sampling and ensemble averaging of regulatory programs, which resulted in more accurate regulator assignments [37].
3 The Lemon-Tree software suite for module network inference

3.1 Lemon-Tree software package

Lemon-Tree is a software suite implementing all of the algorithms discussed in Section 2.2. Lemon-Tree has been benchmarked using large-scale tumor datasets and shown to compare favorably with other module network inference methods [15]. Its performance has been carefully assessed also in an independent study not involving the software authors [42]. Lemon-Tree is self-contained, with no external program dependencies, and is entirely coded in the Java™ programming language. Users can download a pre-compiled version of the software, or alternatively they can download and compile the software from the source code, which is available on the GitHub repository (https://github.com/eb00/lemon-tree). Note that there is also a complete wiki on the Lemon-Tree GitHub (https://github.com/eb00/lemon-tree/wiki), with detailed instruction on how to download, compile, use the software, what are the default parameters and an extended bibliography on the topic of module networks.

Lemon-Tree is a command-line software, with no associated graphical user interface at the moment. The different steps for building the module network are done by launching commands with different input files that will generate different output files. All the command line examples below are taken from the Lemon-Tree tutorial, that users are encouraged to download and reproduce by themselves.

The purpose of the Lemon-Tree software package is to create a module network from different types of ‘omics’ data. The end result is a set of gene clusters (co-expressed genes), and their associated “regulators”. The regulators can be of different types, for instance mRNA expression, copy-number profiles, variants (such as single nucleotide variants) or even clinical parameter profiles can be used. There are three fundamental steps or tasks to build a module network with Lemon-Tree (Figure 2):

- Generate several cluster solutions (“ganesh” task).
- Merge the different cluster solutions using the fuzzy clustering algorithm (“tight_clusters” task).
- Assign regulators to each cluster, producing the module network (“regulators” task).

3.2 Ganesh task

The goal of this task is to cluster genes from a matrix (rows) using a probabilistic algorithm (Gibbs sampling) [41]. This step is usually done on the mRNA expression data only, although some other data type could be used, for instance proteomic expression profiles. We first select genes having non-flat profiles, by keeping genes having a standard deviation above a certain value (0.5 is often used as the cutoff
Define a biological question, e.g. influence of copy-number alterations on co-expressed gene modules in glioblastoma cancer.

Preprocess expression data matrix (sample and gene selection, normalization)

Select candidate regulator types (gene expression, microRNA, copy-number profiles, epigenetic profiles, SNPs, etc.). Preprocess input data.

Infer co-expressed gene clusters from expression data matrix ganesh

Build consensus modules of co-expressed genes. tight_clusters

Infer an ensemble of regulatory programs for a set of co-expressed gene clusters and compute a consensus score (i.e. build the module network) regulators

Draw publication-ready figures for modules figures

Calculate gene ontology (GO) enrichment for each module. go_annotation

Biological interpretation and analysis (pathways, gene hubs, etc...)

Fig. 2 Flow chart for module network inference with Lemon-Tree. This figure shows the general workflow for a typical integrative module network inference with Lemon-Tree. Blue boxes indicate the pre-processing steps that are done using third-party software such as R or user-defined scripts. Green boxes indicates the core module network inference steps done with the Lemon-Tree software package. Typical post-processing tasks (orange boxes), such as GO enrichment calculations, can be performed with Lemon-Tree or other tools. The Lemon-Tree task names are indicated in red (see main text for more details). Figure reproduced from [15] under Creative Commons Attribution License.

score, but this value might depend on the dataset). The data is then centered and scaled (by row) to have a mean of 0 and a standard deviation of 1. To find one clustering solution, the following command can be used (the command is spread here over multiple lines, but should be entered on a single line without the backslash characters):

```java
java -jar lemontree.jar -task ganesh -data_file data/expr_matrix.txt -output_file ganesh_results/cluster1
```

The clustering procedure should be repeated multiple times, using the same command, only changing the name of the output file. For instance we could generate 5 runs, named cluster1, cluster2, cluster3, cluster4 and cluster5, with the same command, just by changing the name of the output file.
3.3 Tight clusters task

Here, we are going to generate a single, robust clustering solution from all the individual solutions generated at the previous step, using a graph clustering algorithm [43]. Basically, we group together genes that frequently co-occur in all the solutions. Genes that are not strongly associated to a given cluster will be eliminated.

```
java -jar lemontree.jar -task tight_clusters \
-data_file data/expr_matrix.txt \
-cluster_file cluster_file_list \
-output_file tight_clusters.txt \
-node_clustering true
```

The “cluster file” is a simple text file, listing the location of all the individual cluster files generated at the “ganesh” step. By default, the tight clusters procedure is keeping only clusters that have a minimum of 10 genes (this can be easily changed by overriding a parameter in the command).

3.4 Revamp task

This task is aimed at maximizing the Bayesian coexpression clustering score of an existing module network while preserving the initial number of clusters. A threshold can be specified to avoid that genes are reassigned if the score gain is below this threshold and allowing the systematic tracking of the conservation and divergence of modules with respect to the initial partition. This task can be used to optimize an existing module network obtained with a different clustering algorithm, or to optimize an existing module network for a different data matrix, e.g. a subset of samples as presented in Section 4.

```
java -jar lemontree.jar -task revamp \
-data_file data/expr_matrix.txt \
-cluster_file cluster_file.txt \
-reassign_thr 0.0 \
-output_file revamped_clusters.txt \
-node_clustering true
```

The “cluster_file.txt” is a simple text clustering file, like the one obtained in “tight_clusters” step, and “reassign_thr” is the score gain threshold that must be reached to move a gene from one cluster to another. By default, this reassignment threshold is set to 0.
3.5 Regulators task

In this task, we assign sets of “regulators” to each of the modules using a probabilistic scoring, taking into account the profile of the candidate regulator and how well it matches the profiles of co-expressed genes [37]. The candidate regulators can be divided into two different types, depending on the nature of their profiles: continuous or discrete. The first type can be for example transcription factors or signal transducers mRNA expression profiles (selected from the same matrix used for detecting co-expressed genes), microRNA expression profiles or gene copy-number variants profiles (CNVs). For the latter, the numerical values will be integers, such as the different clinical grades characterizing a disease state (discrete values), or single nucleotides variants profiles (SNVs, characterized by profiles with 0/1 values). In all cases, the candidate regulator profiles must have been made on the same samples as the tight clusters defined previously. Missing values are allowed, but obviously they should not constitute the majority of the values in the profile. Note that a patch to the regulator assignment implementation identified in [42] is included in LemonTree version 3.0.5 or above.

Once the list of candidate regulators is established, the assignment to the clusters can be made with a single command like this:

```
java -jar lemontree.jar -task regulators
-data_file data/expr_matrix.txt \ 
-reg_file data/reg_list.txt \ 
-cluster_file tight_clusters.txt \ 
-output_file results/reg_tf
```

The “reg_file” option is a simple text list of candidate regulators that are present in the expression matrix. If the regulators are discrete, it is mandatory to add a second column in the text file, describing the type of the regulator (“c” for continuous or “d” for discrete). The profiles for co-expressed genes and for all the regulators must be included in the matrix indicated by the data_file parameter.

Note that this command will create four different output files, using the “output_file” parameter as the prefix for all the files.

- `reg_tf.topreg.txt`: Top 1% regulators assigned to the modules.
- `reg_tf.allreg.txt`: All the regulators assigned.
- `reg_tf.randomreg.txt`: Regulators assigned randomly to the modules.
- `reg_tf.xml.gz`: xml file containing all the regulatory trees used for assigning the regulators.

The regulators text files all have the same format: three columns representing respectively the regulator name, the module number and the score value.
3.6 Figures task

This task is creating one figure per module. The figure represent the expression values color-coded with a gradient ranging from dark blue (low expression values) to bright yellow (high expression values). All the module genes are in the lower panel while the top regulators for the different classes or types of regulators (if any) are displayed in the upper panel. A regulation trees is represented on top of the figure, with the different split points highlighted on the figure as vertical red lines. The name of each gene is displayed on the left of the figure.

```java
java -jar lemontree.jar \
-task figures \ 
-top_regulators reg_files.txt \ 
-data_file data/all.txt \ 
-reg_file data/reg_list.txt \ 
-cluster_file tight_clusters.txt \ 
-tree_file results/reg_tf.xml.gz
```

Note that the “top_regulators” parameter is a simple text file listing the different top regulator files and their associated clusters. Such a file could be for instance the file reg_tf.topreg.txt mentionned in the previous paragraph. All figures are generated to the eps (encapsulated postcript) format, but it is relatively easy to convert this format to other common formats such as pdf.

3.7 GO annotation task

The goal of this task is to calculate the GO (Gene Ontology) category enrichment for each module, using code from the BiNGO package [44]. We have to specify two GO annotation files that are describing the GO codes associated with the genes (“gene_association.goa_human”) and another file describing the GO graph (“gene_ontology_ext.obo”). These files can be downloaded for various organisms from the GO website (http://www.geneontology.org). We also specify the set of genes that should be used as the reference for the calculation of the statistics, in this case the list of all the genes that are present on the microarray chip (file “all_gene_list”). The results are stored in the output file “go.txt”.

```java
java -jar lemontree.jar \
-task go_annotation \ 
-cluster_file tight_clusters.txt \ 
-go_annot_file gene_association.goa_human \ 
-go_ontology_file gene_ontology_ext.obo \ 
-go_ref_file all_gene_list \ 
-output_file go.txt
```
4 Differential module network inference

4.1 Differential module network model

Assume that we have expression data in $T$ different conditions (e.g., experimental treatments, cell or tissue types, disease stages or states), with $N_t$ samples in each condition $t \in \{1, \ldots, T\}$, and wish to study how the gene regulatory network differs (or not) between conditions. We define a differential module network as a collection of module networks $\{M_1, \ldots, M_T\}$, one for each condition, subject to constraints, and model gene expression levels for $G$ genes as

$$p(x_1, \ldots, x_G \mid M_1, \ldots, M_T) = \prod_{t=1}^{T} p(x_1, \ldots, x_G \mid M_t), \quad (4)$$

where each factor is a model of the form of eq. (2). Hence, if gene $i$ is assigned to modules $\{k_{i1}, \ldots, k_{iT}\}$ in each module network, its parent set is the union $\mathcal{P}_i = \bigcup_{t=1}^{T} \mathcal{P}_{ki}$. If the graph mapping these parent sets to their targets is acyclic, eq. (4) defines a proper Bayesian network. If the the individual factors $p(x_1, \ldots, x_G \mid M_t)$ are the usual Gaussians with parameters depending on the parent expression levels in that module network, their product remains a Gaussian. By Bayes’ theorem we can write, for a concatenated data matrix $X = (X_1, \ldots, X_T)$,

$$p(M_1, \ldots, M_T \mid X) \propto p(M_1, \ldots, M_T) \prod_{t=1}^{T} p(X_t \mid M_t) \quad (5)$$

If we assume independence, $p(M_1, \ldots, M_T) = \prod_t p(M_t)$, then optimization of, or sampling from, eq. (5), is the same as inferring module networks independently in each condition, but this will reveal little of the underlying relations between the conditions. Instead we assume that there exists a conserved set of modules across all conditions, but their gene and regulator assignment may differ in each condition. This results in the following constraints:

1. The number of modules must be the same in each module network, i.e.
   $$p(M_1, \ldots, M_T) = 0 \text{ unless } K_1 = \cdots = K_T = K.$$
2. Module networks with more similar gene and/or regulator assignments are more likely a priori,
   $$\log p(M_1, \ldots, M_T) = -\sum_{k=1}^{K} \sum_{t \neq t'} \left[ \lambda_{k,t,t'} f(\mathcal{P}_k(t), \mathcal{P}_k(t')) + \mu_{t,t'} g(\mathcal{P}_k(t), \mathcal{P}_k(t')) \right], \quad (6)$$

where $f$ and $g$ are distance functions on sets (e.g. Jaccard distance) and $\lambda_{k,t,t'}$ and $\mu_{t,t'}$ are penalty parameters that encode the relative a priori similarity between conditions.
4.2 Optimization algorithm

For simplicity we assume here that $\mu_{t,t'} = 0$ and $\lambda_{t,t'} = \lambda$ for all $(t,t')$ in eq. (6), i.e. we will only constrain the gene assignments, and uniformly so for all condition pairs; the complete model will be treated in detail elsewhere. More general forms of $\lambda_{t,t'}$ can be used for instance to mimic the model of [36], where conditions represented different species and gene reassignments were constrained by a phylogenetic tree. With a fixed $\lambda$, instead of modelling $\lambda$ and $f$ explicitly, we observe that the effect of including $f$ in the model (5) is to impose a penalty on gene reassignments: starting from identical modules in all conditions, a gene reassignment in condition $t$ increases the posterior log-likelihood only if its increase in $\log p(X_t | M_t)$ is sufficiently large to overcome the penalty induced by eq. (6). This can be modelled equivalently by setting a uniform module reassignment score threshold as an external parameter. Hence we propose the following heuristic optimization algorithm for differential module network inference using Lemon-Tree:

1. Create a concatenated gene expression matrix $X = (X_1, \ldots, X_T)$ and learn a set of coexpression modules using tasks “ganesh” (Section 3.2) and “tight clusters” (Section 3.3). This results in a set of module networks $(M_1, \ldots, M_T)$ with identical module assignments and empty parent sets.

2. Set a reassignment threshold value and use task “revamp” (Section 3.4) to maximize the Bayesian coexpression clustering score $\log p(X_t | M_t)$ [cf. eq. (3)] for each condition independently, but subject to the constraint that gene reassignments must pass the Bayesian score difference threshold.

3. Assign regulators to each module for each condition independently using task “regulators” (Section 3.5).

4.3 Reconstruction of a differential module network between atherosclerotic and non-atherosclerotic arteries in cardiovascular disease patients

To illustrate the differential module network inference algorithm, we applied it to 68 atherosclerotic (i.e. diseased) arterial wall (AAW) samples and 79 non-atherosclerotic (i.e. non-diseased) internal mammary artery (IMA) samples from the Stockholm Atherosclerosis Gene Expression study [45–47], using 1803 genes with variance greater than 0.5 in the concatenated data. The STAGE study was designed to study the effect of genetic variation on tissue-specific gene expression in cardiovascular disease [46]. According to the systems genetics paradigm, genetic variants in regulatory regions affect nearby gene expression (“cis-eQTL effects”), which then causes variation in downstream gene networks (“trans-eQTL effects”) and clinical phenotypes [47,48]. We therefore considered as candidate regulators the tissue-specific sets of genes with significant eQTLs [46] and present in our filtered
gene list (668 AAW and 964 IMA genes, 267 in common), and ran the “regulators”
task on each set of modules independently.

As expected, independent clustering of the two data sets results in different num-
bers of modules, and an inability to map modules unambiguously across tissues
(Figure 3a). In contrast, application of the differential module network optimiza-
tion algorithm (Section 4.2) results in a one-to-one mapping of modules, whose
average overlap varies smoothly as a function of the reassignment threshold value
(Figure 3b).

The biological assumption underpinning the differential module network model
(Section 4.1) is that each module represents a higher-level biological process, or
set of processes, that is shared between conditions, whereas the differences in
gene assignments reflect differences in molecular pathways that are affected by,
or interact with, this higher-level process. To test whether the optimization algo-

rithm accurately captures this biological picture, we first performed gene ontol-
ogy enrichment (task “go_annotation”, Section 3.7) using the GO Slim ontology.
GO Slims give a broad overview of the ontology content without the detail of
the specific fine-grained terms (http://www.geneontology.org/page/
go-slim-and-subset-guide). Consistent with our biological assumption,
mapping modules in atherosclerotic and non-atherosclerotic tissue are often en-
riched for the same GO Slim categories (Figure 4).

Next, we performed gene ontology enrichment using the complete, fine-grained
ontology, and removed all enrichments that were shared between matching modules.
The resulting tissue-specific module enrichments reflected biologically meaningful
Fig. 4 Enrichment for GO Slim terms in the STAGE AAW-IMA differential module network. Blue nodes are modules, yellow nodes GO Slim terms. Red and green edges indicate enrichment ($q < 0.05$) in the corresponding AAW and IMA module, respectively. The reassignment threshold used is 0.015.

In summary, this application has shown that differential module network inference allows to identify sets of one-to-one mapping modules representing broad biological processes conserved between conditions, with biologically relevant differences in fine-grained gene-to-module assignments and upstream regulatory factors.
Reassignment of nodes was computed with a threshold of 0.015. Blue nodes are modules, yellow nodes GO terms, grey nodes regulatory genes. Red and green edges indicate tissue-specific enrichment ($q < 0.01$) in the corresponding AAW and IMA module, respectively. Dashed red and green edges indicate regulator assignments in AAW and IMA, respectively. Only top 1% regulators are depicted. neg., negative; pos., positive; pr., process; reg., regulation; resp., response.
Acknowledgements  PE and TM are supported by Roslin Institute Strategic Programme funding from the BBSRC [BB/P013732/1].

References


36. Sushmita Roy, Ilan Wapinski, Jenna Pfiffner, Courtney French, Amanda Socha, Jay Konieczka, Naomi Habib, Manolis Kellis, Dawn Thompson, and Aviv Regev. Arboretum:


