Microarray Data Analysis and Pathway Activity Inference in PATIKA

Özgün Babur §, Emek Demir §, Aslı Ayaz, Uğur Doğrusöz §, † and Onur Sakarya

Center for Bioinformatics, Computer Engineering Department, Bilkent University, Ankara 06800, Turkey

§ Presenting authors † Corresponding author

Abstract

PATIKA is a project that aims to develop methods and software tools for effective analysis of complex biological data at a functional level. Pathway editor of PATIKA facilitates construction, storage, integration and analysis of static pathway graphs.

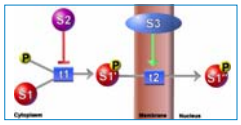
The microarray analysis component of PATIKA helps to filter expression data and display on the related pathways. Moreover, the editor infers the possible differential activity on the network based on the loaded expression data, making use of the embedded algorithms.

Ontology

PATIKA is built on a state level ontology for an intuitive, comprehensive, uncomplicated representation of cellular events.

States are the actors of the cellular events. They are either macromolecules (e.g., DNAs, RNAs, and proteins), small molecules (e.g., ions), or physical events (e.g., heat, radiation).

Transitions are the changes that states undergo.

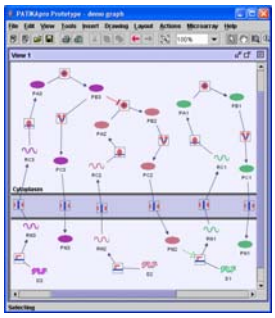


Interactions are the relations of states with transitions such as substrate, product, activator and inhibitor.

Cellular ***compartments*** are also modeled with PATIKA .

Software Framework

Static Pathway Graph



A hypothetical pathway model constructed with PATIKA editor. Note that static pathway models do not capture the temporal relations, such as the actual flow of a signal, as they model possible routes, but not the event itself.

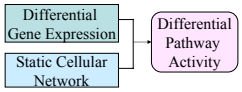
Microarray Data

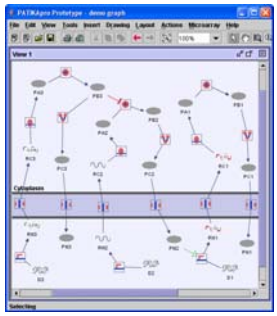
Microarray data can be loaded to the editor using an optional filter to filter out insignificant and/or untrusted data, which is visualized by color coding and/or by the labels of the corresponding RNA states.

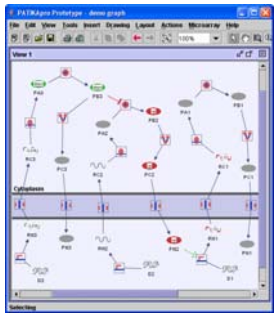
the filter dialog of the editor

Activity Inference

Pathway activity inference is the problem of propagating this activity status to the other states of the pathway.







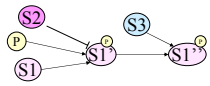
An embedded algorithm in PATIKA editor determines all possible dependencies between differentially expressed genes. The algorithm makes use of the regulation paths between the RNA states. Those microararay data compatible regulation paths are the possible links between gene expressions. The activity inference on the network is also visualized by color coding and labeling.

Algorithm

Our algorithm is based on a depth first traversal (DFT), whose source and target node sets are differentially expressed RNA states.

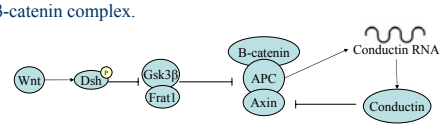
Regulation Graph

Still preserving our ontology, for illustration purposes we use regulation graphs, in which we only show up and down regulation relations between states. For instance the regulation graph version of the PATIKA graph in ontology section is shown below.



Cyclic Regulations

We do not need to evaluate the regulation paths that contain cyclic relations. For instance in the figure below there is a cyclic down-regulation path from Wnt to β-catenin complex, but this is semantically wrong since activity of this path also up-regulates the β-catenin complex.



Definitions

A regulation path is *compatible* with a given microarray data if:

→ the path is positive and the activity of its start and end nodes changed in the same direction *or*

→the path is negative and the activity of its start and end nodes changed in the reverse direction.

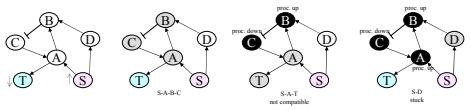
The path is called *conflicting* with the expression data otherwise.

Note that the compatibility of the path is only considered when both of its start and end points are significant for the given expression profile.

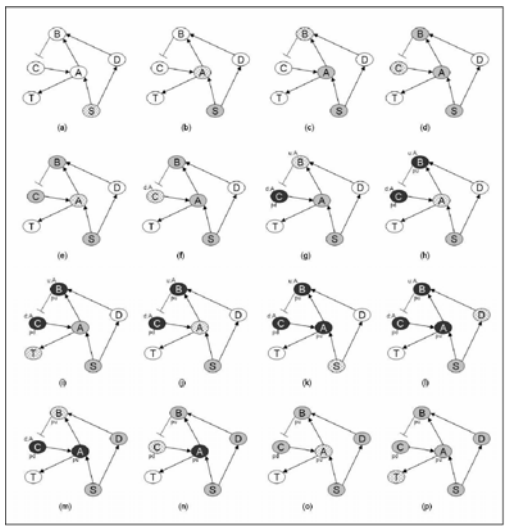
Modifications to DFT

In classic DFT, there is a single label for node status, but we need two labels per node since there are two kinds of regulation types (up and down) of paths and each kind needs to be evaluated separately.

Even with two status labels, we do not guarantee to find all possible regulation paths. For example in the drawing below the traversal ends on the S-D-B path because B was already processed with up-regulation.



In addition to the labels, we maintain two lists per node. These lists contain the nodes that caused a cycle during the traversal, for up-regulation and down-regulation. Using these lists, some processed nodes are re-processed. The example execution of the algorithm on a sample graph is illustrated below.



(a) CN (current node): S, CP (current path) is empty.

(b) CN: A, CP: S.

(c) CN: B, CP: S => A.

(d) CN: C, CP: S => A => B.

(e) CN: A, CP: S => A => B => C. A cycle is detected. We turn back. Node A returns itself in a set.

(f) CN: C, CP: S => A => B => C. Node C receives returned set of A and adds it to its down-regulation cycle set.

(g) CN: B, CP: S => A => B. Node C is marked as processed for down-regulation and returns its down-regulation cycle set to B. B adds the items in the returned set to its up-regulation cycle set.

(h) CN: A, CP: S => A. B is processed for up-regulation and returned its up-regulation cycle set to A. A removes itself from the returned cycle set.

(i) CN: T, CP: S => A. S => A => T path is an up-regulation path and conflicts with the expression data, thus it is ignored.

(j) CN: A, CP: S => A. T has not been processed yet.

(k) CN: S, CP: S.

(l) CN: D, CP: S.

(m) CN: B, CP: S => D. Up-regulation cycle set of B contains A, and A is not in the current path. So A is removed from the cycle set of B and B is re-traversed.

(n) CN: C, CP: S => D => B. Down-regulation cycle set of C contains A, which is not in current path; so A is removed from the cycle set of C and C is re-traversed.

(o) CN: A, CP: S => D => B => C. A is not processed for down-regulation, therefore it is traversed.

(p) CN: T, CP: S => D => B => C => A. Voila! A regulation path that is compatible with the expression data is found.