## LemonTree algorithm improvement

1. Overview of the issue in regulators assignment

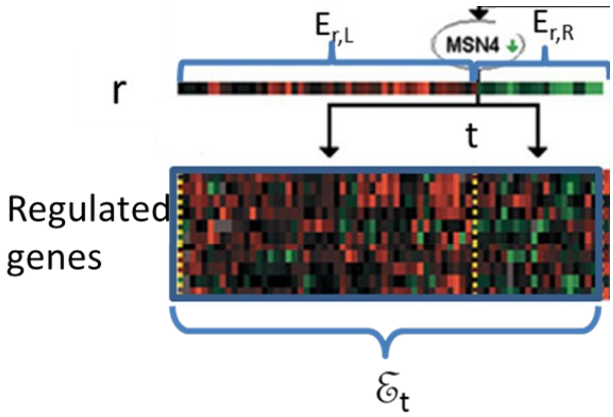


**Fig S5-1. An illustration of the "opposite expressions" pattern where the issue occurs (adapted from(Segal *et al.*, 2003)).**

Given a binary tree consisting of two branches *L*, *R* and a split *t*, under which experiments (i.e. conditions) $E_r$ are partitioned into $E_{r,L}$ and $E_{r,R}$, *r* denotes a candidate regulator, and $E_r$ its expression values under the split *t*; $E_{r,L}$, $E_{r,R}$ denote its expression values in the left and right branch of *t*, respectively (Fig S5-1). The expressions of $E_{r,L}$, $E_{r,R}$ are called *opposite* when

$$E_L < E_R, \forall E_L \in E_{r,L}, \forall E_R \in E_{r,R} \; or \; E_L > E_R, \forall E_L \in E_{r,L}, \forall E_R \in E_{r,R} \quad (1)$$

During LemonTree's *regulators assignment* task, candidate regulators with distinctly *opposite expressions* at the left and right branches of a given split and hence a perfect predictability of the regulated genes, can be excluded from assignment, eventually leading to a significant loss of highly likely regulators as well as erroneously prioritization of the less likely ones.

Below is an example of such situation and the issue in regulators assignment.

Consider a module that contains the expressions of the regulated genes from two experimental conditions (GRP1, GRP2), with two replicates (R1, R2) per condition. The regulated genes have similar expressions within each condition but great difference between conditions (Fig S5-2A).

Suppose the candidate regulators consist of two sets of genes (total number 20), the former (REG_1 to REG_10) has opposite expressions as defined above, while the latter (REG_11 to REG_20) does not (Fig S5-2B). Theoretically, the former set is more likely to contain true regulators than the latter.

The *regulators assignment* step by LemonTree (v3.0.2) assigned regulators to this module from the 20 candidates and the results (the expression heatmaps of the

regulated genes and of all the assigned regulators) are shown in Fig S5-3. However, none of the first set of candidates (REG_1 to REG_10) are assigned as regulators.

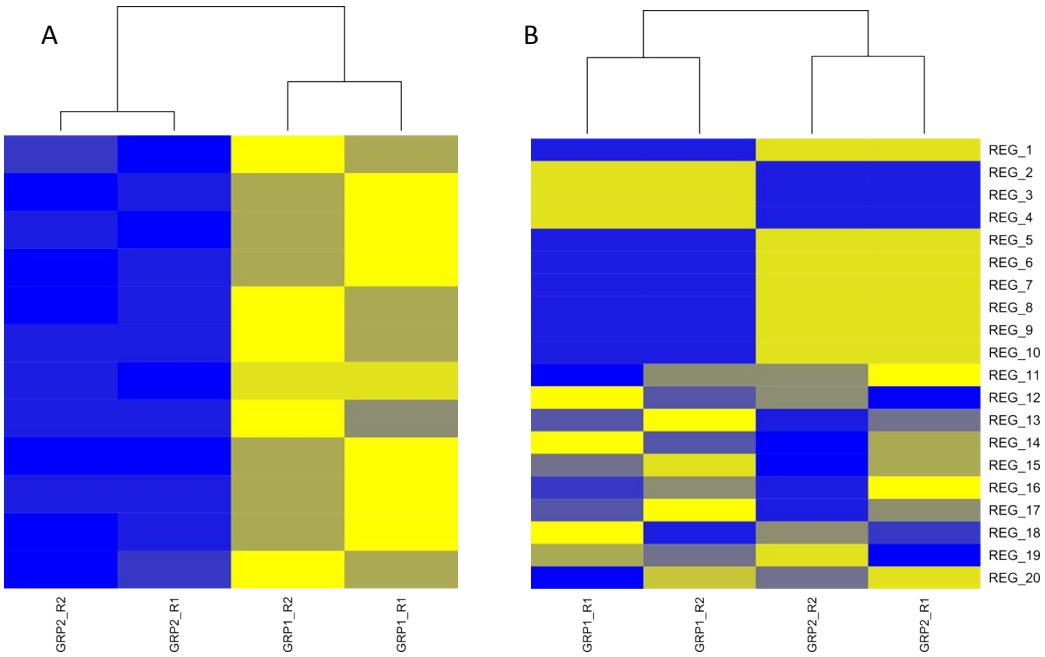


**Fig S5-2. Expression heatmap of the regulated genes (A) and candidate regulators (B).**

## 2. The cause in algorithm

The aforementioned issue is caused by the bisection method in search of the optimal $\beta$ that maximizes the posterior probability of a regulator's expressions to predict the partition of conditions at a given split: the algorithm discards candidates whose posterior probability increases monotonically with $\beta$ and plateaus at 1 when $\beta \to +\infty$. Candidates with *opposite* expressions are just such case.

The posterior probability a candidate *r* regulates at split *t* given its expression $x_{r_t}$ is

$$\Pr\left(y_t \middle| x_{r_t}, z_t, \beta_t\right) \propto \max_{\beta} \prod_{m \in E_t} Pr(y_{t,m} | x_{r_{t,m}}, z_t, \beta_t) \tag{2}$$

where $\Pr\left(y_{t,m} \middle| x_{r_{t,m}}, z_t, \beta_t\right) = 1/(1 + e^{-\beta_t y_t (x_{r_t} - x_t)})$, $z_t$ is the split value which partitions the expressions of *r*, $y_t$ is a binary variable ($y_t = -1$ means the left branch, $y_t = 1$ the right branch), $\beta$ is the parameter that specifies the fuzziness of the decision tree: when $\beta = 0$, the probability that a candidate *r* regulates at split is always 1/2 regardless its expression; while $\beta \to +\infty$, it becomes *hard* decision tree ($\Pr\left(y = +1 \middle| x_{r_t} > z_t\right) = 1, \Pr\left(y = -1 \middle| x_{r_t} < z_t\right) = 1$).

$$f(\beta) = log\max_{\beta} \prod_{m \in E_t} Pr(y_{t,m}|x_{r_{t,m}},z_t,\beta_t)$$

Let , therefore

$$\beta = arg\max_{\beta} f(\beta) = \operatorname{argmax}_{\beta} \{ - \sum_{m \in E_{t,L}} \log\left(1 + e^{-y_t\beta\left(x_{r_t,m} - z_t\right)}\right) - \sum_{m \in E_{t,R}} \log\left(1 + e^{y_t\beta\left(x_{r_t,m} - z_t\right)}\right)\}$$

(3)

If the function $f(\beta)$ is convex upward, then $\beta$ should be a root of

$$f'(\beta) = y_t\left\{\sum_{m \in E_{t,L}} \frac{x_{r_t,m} - z_t}{1 + e^{y_t\beta\left(x_{r_t,m} - z_t\right)}} - \sum_{m \in E_{t,R}} \frac{x_{r_t,m} - z_t}{1 + e^{-y_t\beta\left(x_{r_t,m} - z_t\right)}}\right\} = 0 \quad (4)$$

LemonTree (v3.0.2) uses a bisection method to find the root with respect to Equation (4).

However, here it is easy to prove that there is a situation in which $f(\beta)$ monotonically increases and reaches plateau at $\beta = +\infty$.

Suppose that

$$\begin{cases} x_{r_{t,m}} = x_L, \forall m \in E_L, \\ x_{x_{t,m}} = x_R, \forall m \in E_R, \\ x_L < x_R \end{cases} \quad (5)$$

So that $y_t = -1$ (namely, the left leaf implies regulator's down-regulation, the right up-regulation). Let $z_t = 1/2(x_L + x_R)$, $\delta = 1/2(x_R - x_L)$. Let $|E_L| = |E_R| = k$. Hence

$$f'(\beta) = 2k\frac{\delta}{1 + e^{\beta\delta}} > 0, \forall\, \beta > 0$$

LemonTree's bisection procedure goes as follows. Initially it sets the searching range of $\beta$ to be (0, 20), then it checks the signs of $f'(\beta)$ at the boundaries. If they are opposite, suggesting the root lies within and the program halves the range (keeping the signs at new boundaries opposite) and continues; if they are the same, suggesting the root is beyond the right-hand side boundary, the program expands the right-hand side boundary by a factor of 2 and repeats the sign check step. When it detects that the signs of $f'(\beta)$ can never be opposite after the maximum times of range expansion, it simply discards such candidate *r*.

The candidate whose expressions satisfy condition 5 obviously shows regulatory potential, but is excluded from assignment due to the fact that the bisection procedure can never reach a negative $f'(\beta)$ for any $\beta > 0$.

## 3. A possible workaround

Having noticed that $\beta \to +\infty, f'(\beta) \to +0$, the maximum $f(\beta)$ can be approximated when $\beta$ is truncated at a value large enough. Thus we can enable the range expansion procedure to return the right-hand side boundary $\beta_{max}$ such that $f'(\beta_{max}) \approx 0$. Then the bisection will obtain an approximated $\beta$ maximizing $f(\beta)$, that is, actually the right-hand side boundary.

We patched the algorithm accordingly, and tested it against the same data. As shown in Fig S5-4, our workaround indeed restores the identification of missing candidates as top regulators.

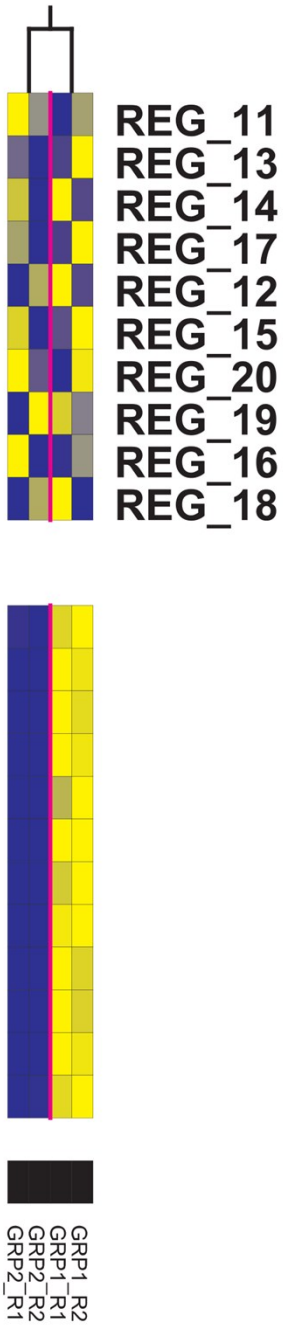

**Fig S5-3. Expression heatmaps of the regulated genes and regulators.** Bottom panel: the regulated genes; top panel: the assigned regulators using the original LemonTree version v3.0.2.

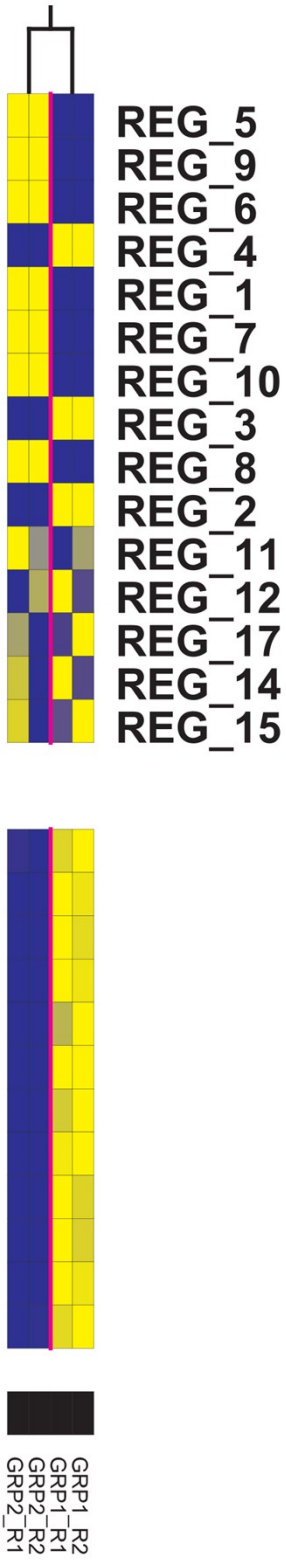


**Fig S5-4. Expression heatmaps of the regulated genes and regulators.** Bottom panel: the regulated genes; top panel: the assigned regulators using the patched algorithm.

4. The code patch

The patch applied to the source code in `src/lemontree/utils/RootFinder.java` is

```
  1 84,85c84,85
  2 <       if (f*fmid >= 0.0)
  3 <               throw new Exception("Root must be bracketed for
bisection.");
  4 ---
  5 >       //if (f*fmid >= 0.0)
  6 >           //   throw new Exception("Root must be bracketed for
bisection.");
  7 120c120
  8 <           if (f1*f2 < 0)
  9 ---
 10 >           if ((f1 * f2) < 0 || (Math.abs(f2) < xAcc))
```

## Reference


Segal,E. *et al.* (2003) Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat. Genet.*, **34**, 166–176.