Optimization of subnetwork statistics

Benno Schwikowski

March 19, 2014

1 "Active Modules" statistic

The Active Modules statistic (Ideker et al., 2002) is based on the following. Given a series of sorted p-values $p_1, \cdots, p_m$, $j^* = \arg\min_j p_{A(j)}$ has to be calculated, where

$$p_{A(j)} = \sum_{h=j}^{m} p_{j,h}, \text{ where } p_{j,h} = \sum_{h=j}^{m} \binom{m}{h} p_j^h (1-p_j)^{m-h}.$$

2 Relationship between successive sum terms $p_{j,h}$ and $p_{j,h+1}$

One set of optimizations are based on the observation that, for successive sum terms $p_{j,h}$ and $p_{j,h+1}$:

$$\frac{p_{j,h+1}}{p_{j,h}} = \frac{\frac{m!}{(m-(h+1))!\cdot(h+1)!} \cdot p_j^{h+1}(1-p_j)^{m-(h+1)}}{\frac{m!}{(m-h)!\cdot h!} \cdot p_j^h (1-p_j)^{m-h}} = \frac{p_j \cdot (m-h)}{(1-p_j)\cdot(h+1)},$$

i.e.,

$$p_{j,h+1} = p_{j,h} \cdot \frac{p_j \cdot (m-h)}{(1-p_j)\cdot(h+1)}, \tag{1}$$

which can be written as $p_{j,h+1} = p_{j,h} \cdot f_{m,p_j}(h)$, with $f_{m,p}(h) = \frac{p\cdot(m-h)}{(1-p)\cdot(h+1)}$. Note that $f_{m,p}(h)$ is a decreasing function in h, i.e.,

$$f_{m,p}(h) \leq f_{m,p}(h') \Longleftrightarrow h \leq h'. \tag{2}$$

3 Quick determination of a dominating term p_{j,h^*}

Using equations (**??**) and (**??**), determining a *dominating* term p_{j,h^*} ($p_{j,h^*} \geq p_{j,h}$ for all h) is useful later optimizations. As

$$\begin{aligned} f_{m,p}(h) < 1 &\iff p \cdot (m-h) < (1-p) \cdot (h+1) \\ &\iff p \cdot m < h - p + 1 \\ &\iff p \cdot (m+1) - 1 < h, \end{aligned}$$

h^* can be chosen to be the smallest integer larger than $p \cdot (m+1) - 1$, (or $h^* = j$ if $p \cdot (m+1) - 1 < j$). It is then easy to verify that p_{j,h^*} is indeed dominating.

Optimization 1. *If many dominating terms are to be calculated, multiple expensive computations of the same binomial coefficients $\binom{m}{h}$ can be avoided by using a hash that stores previously computed values of $\binom{m}{h}$.*

4 Quick elimination of irrelevant terms $p_{j,h}$

The two sequences $p^+ = p_{j,h^*+1}, p_{j,h^*+2}, \cdots$ and $p^- = p_{j,h^*-1}, p_{j,h^*-2}, \cdots$ decrease more than exponentially (per Equations (**??**) and (**??**); of course, terms $p_{j,h}$ are only defined for $h = j, ..., m$). Thus, only a limited number of the first terms of both sequences need to be evaluated after computing p_{j,h^*+1}, to determine $p_{A(j)} = \sum_{h=j}^{m} p_{j,h}$ up to a given accuracy.

Optimization 2. *p^+ and p^- need to be evaluated only up to the point where the next addend, say p_{j,h^*+i}, does not contribute to the numerical approximation of $p_{A(j)}$ at a given accuracy. Whether or not this is the case can be tested by computing $p_{j,h^*+i}/s$, where s is the sum of all previously computed terms $p_{j,h}$, and continuing only if $p_{j,h^*+i}/s > \delta$ (where, e.g., $\delta = 10^{-6}$).*

Optimization 3. *It can be seen from Eq. (**??**) that the last required index i only depends on m, j, and h^*, and can therefore be precomputed in advance, or its value can be hashed.*

Asymptotically, for many calculations of $p_{A(j)}$ this saves an additional multiplication for each computed term $p_{m,h}$.

5 Efficient calculation of other terms $p_{j,h}$

Optimization 4. *Once a single term of the sum (e.g., a dominating term) $p_{j,h}$ has been calculated, neighboring terms $p_{j,h-1}$ and $p_{j,h+1}$ can be computed using just two multiplications:*

$$p_{j,h+1} = p_{j,h} \cdot p^+ \cdot f^+_{m,h},$$

and

$$p_{j,h-1} = p_{j,h} \cdot p^- \cdot f^-_{m,h},$$

where $p^+ = \frac{p}{1-p}$ *and* $f^+_{m,h} = \frac{m-h}{h+1}$, *as well as* $p^- = 1/p^+$ *and* $f^-_{m,h} = 1/f^+_{m,h}$ *can be precomputed for all pairs* (m, h).

6 Avoiding the computation of $p_{A(j)}$ for certain j

After the previous optimizations for the calculation of $p_{A(j)}$ for a given j, we describe here an approach to efficiently calculate $j^* = \arg\min_j p_{A(j)}$.

Not only the elements of the series $p_{A(j)} = \sum_{h=j}^{m} p_{j,h}$ can vary by orders of magnitude, but also the values of $p_{A(j)}$, $j = 1, \cdots, m$, which motivates further optimizations.

Once $p_{A(j)}$ is known for one value of j, it can be used to optimize the calculation of other $p_{A(j')}$, $j' \neq j$.

Optimization 5. *We therefore propose to first calculate only the dominating terms* $p_{j,h^*(j)}$ *for all* j. *One can then compute* $j_0 = \arg\min_j (p_{j,h^*(j)})$, *and* $p_{A(j_0)}$. *Then, for those* $j \neq j_0$ *with* $p_{j,h^*(j)} > p_{A(j_0)}$, $p_{A(j)}$ *does not need to be computed, as* $p_{A(j)} \geq p_{j,h^*(j)} > p_{A(j_0)}$, *and, therefore,* $j \neq j^*$.

7 Minimizing the cost of sorting p-values

In the course of the simulated annealing approach proposed by Ideker et al. (2002), the statistic $\min \arg_j(p_{A(j)})$ needs to be computed for many sets of p-values from the connected components of a graph $G = (V, E)$, where V are the nodes in the "on" state of the larger graph $\hat{G} = (\hat{V}, \hat{E})$, and E are the edges in $\hat{E}$ that go between nodes of V. In each step of the Simulated Annealing, V only changes by a single element, i.e., a single node $v \in \hat{V}$ is added to V or removed from V. Consequently, the connected components of the resulting new graph $G' = (V', E')$ are very similar to the known connected components $C_1, \cdots, C_k$ of $G = (V, E)$, and can be found very efficiently by updating $C_1, \cdots, C_k$.

Optimization 6. *Specifically, the sets of concerned p-values change in a single step only in a few, defined, ways, that allow to utilize the sorted lists of p-values from the previous step to avoid sorting. As the state of only a single node* v *is changed, only a single connected component* $C_i \in C_1, \cdots, C_k$, *and its associated lists* L_i *of p-values changes.*

Two cases can be distinguished: [Case 1.] v is switched from "on" to "off". In this case, the p-value p_v can just be removed from L ($(O(1)$ in a linked list implementation). Then, connected components have to be updated (see the optimizations below.) If the removal of v from G breaks C generates several new connected components, the corresponding sublists of p-values can be extracted as sorted sublists of L (O($|C|$)). [Case 2.] v is switched from "off" to "on". If the "on" neighbors of v in $\hat{G}$ belong to the same connected component C_i of G,

v can just be added to C_i, and inserted into the sorted list L_i using insertion sort. In the case of different connected components, these components, and their sorted sublists need to be merged into a single sorted list using merge sort, before p_v is inserted ($O(|C|)$).

8 Avoiding computation of connected components

The lists of p-values that need to be scored depend on the connected components of the graph G. Instead of recomputing connected components after an update of G (insertion or deletion of a node v), these connected components can be more efficiently updated.

Any connected component of G that does not contain a neighbor of v does not need to be updated. This leads to the following optimization

Optimization 7. *When recomputing connected components after removal of a node "v", adapt the algorithm (e.g., Tarjan-Hopcroft) to perform depth-first search only from the neighbors of v.*

Recent work allows more far-reaching optimizations:

Optimization 8. *Connected components can be updated in sublinear time (cf. Chan, Partrascu, and Roditty in FOCS 2008, and updates.) As biological graphs are typically sparse, previous work by Henzinger et. al. on edge updates may be more applicable.*

Unfortunately, the required data structures are quite heavy. Finding a good implementation might be the best best for this.