Understand graph and partition enumeration

Enumerating relationship graphs

Relationship graphs as nested partitions

The function `enumerate_RGs` enumerates all relationship graphs (RGs). A relationship graph can be interpreted as a nested partition. Label the n genotypes (one per vertex) as $v_1, \ldots, v_n$. A clonal partition $\boldsymbol{p}^{\mathrm{cl}} = \{\boldsymbol{c}_1^{\mathrm{cl}}, \ldots, \boldsymbol{c}_{n^{\mathrm{cl}}}^{\mathrm{cl}}\}$ is a partition of $\{v_1, \ldots, v_n\}$ where two genotypes are in the same cell if they have a clonal relationship, and where n^{cl} is the number of cells of the clonal partition (number of clonal cells). A relationship graph can be represented as a partition of $\boldsymbol{p}^{\mathrm{cl}}$, where two clonal cells are in the same subset if they have a sibling relationship. So to enumerate all relationship graphs, we can first enumerate all clonal partitions, and then enumerate the sibling partitions over each clonal partition.

As an example, suppose that we have $n = 4$ genotypes with the following relationships: v_1 and v_3 are clones, v_1 and v_2 are siblings, v_2 and v_3 are siblings, while v_4 is a stranger to all other genotypes. The corresponding clonal partition can be written as

$$\boldsymbol{c}_1^{\mathrm{cl}} = \{v_1, v_3\}, \qquad \boldsymbol{c}_2^{\mathrm{cl}} = \{v_2\}, \qquad \boldsymbol{c}_3^{\mathrm{cl}} = \{v_4\},$$

and the sibling partition is $\{\{\boldsymbol{c}_1^{\mathrm{cl}}, \boldsymbol{c}_2^{\mathrm{cl}}\}, \{\boldsymbol{c}_3^{\mathrm{cl}}\}\}$.

Enumerating clonal partitions

The function `enumerate_CPs` enumerates all clonal partitions (CPs). Each clonal partition can be written as a membership vector $\boldsymbol{q} = (q_1, \ldots, q_n) \in \{1, \ldots, n^{\mathrm{cl}}\}^n$, which indicates that for each $i = 1, \ldots, n$, the genotype v_i belongs to $\boldsymbol{c}_{q_i}^{\mathrm{cl}}$, the q_i-th clonal cell. In order to have a one-to-one correspondence between the membership vectors and the clonal partitions, we require that the first appearance of any positive integer r in $\boldsymbol{q}$ must come after the first appearances of $1, \ldots, r-1$. For the previous example, we have the membership vector $(1, 2, 1, 3)$. A complication about clonal partitions is the fact that genotypes from the same infection cannot belong to the same clonal cell. Let us denote the MOIs as $M_0, \ldots, M_k$. This restriction can can be expressed as

$$\begin{aligned}
q_d &\neq q_{d'} && \text{for any } 1 \leq d < d' \leq M_0, \\
q_{M_0+d} &\neq q_{M_0+d'} && \text{for any } 1 \leq d < d' \leq M_1, \text{ and} \\
q_{M_0+\ldots+M_{t-1}+d} &\neq q_{M_0+\ldots+M_{t-1}+d'} && \text{for any } 1 \leq d < d' \leq M_t \text{ when } t \geq 2.
\end{aligned}$$

We now describe an algorithm to enumerate all clonal partitions by generating all valid membership vectors in lexicographical order. We first initialise $\boldsymbol{q}$ according to

$$\begin{aligned}
q_d &= d && \text{for } d = 1, \ldots, M_0, \\
q_{M_0+d} &= d && \text{for } d = 1, \ldots, M_1, \text{ and} \\
q_{M_0+\ldots+M_{t-1}+d} &= d && \text{for } d = 1, \ldots, M_t \text{ when } t \geq 2,
\end{aligned}$$

which is lexicographically the smallest membership vector. Note that the first M_0 entries of $\boldsymbol{q}$ are fixed due to the restrictions involved. Generating the next membership vector can be broken down into three steps: (i) identifying the shortest suffix that can be changed, (ii) changing the entries of the episode from which the suffix starts from, (iii) replace the remaining part of the suffix with the initial membership vector.

We illustrate these steps with an example where there are $k = 2$ recurrent episodes, $M_0 = 1$, $M_1 = 2$, and $M_2 = 1$. The initial membership vector is $\boldsymbol{q} = (1, 1, 2, 1)$. The shortest suffix that can be changed is to change the final 1 to a 2. In this iteration, steps (ii) and (iii) do not apply, and we get $\boldsymbol{q} = (1, 1, 2, 2)$. The following iteration follows the same logic, which gives $\boldsymbol{q} = (1, 1, 2, 3)$. In the next iteration, we cannot increase the 2 nor the 3, as there are no earlier appearances of those numbers. Step (i) identifies that we have to change the suffix $(1, 2, 3)$. Step (ii)

changes the entries of the first recurrent episode to (2,1), whereas step (iii) replaces the remaining part of the suffix with the initial membership vector $(1, 1, 2, 1)$, i.e. replace the last entry with 1. This results in $\boldsymbol{q} = (1, 2, 1, 1)$. The full list of membership vectors is

$$(1,1,2,1),(1,1,2,2),(1,1,2,3),(1,2,1,1),(1,2,1,2),(1,2,1,3),(1,2,3,1),(1,2,3,2),(1,2,3,3),(1,2,3,4).$$

Here are the steps in detail:

(i) To identify the appropriate suffix, we have to find the rightmost entry of $\boldsymbol{q}$ that has an earlier appearance. This is where the suffix should begin. To find this entry efficiently, it helps to have a prefix maximum vector $\boldsymbol{q}^* = (q_1^*, \ldots, q_n^*)$ where $q_1^* = 0$ and $q_i^* = \max\{q_1, \ldots, q_{i-1}\}$ for $i = 2, \ldots, n$. An interpretation of $\boldsymbol{q}^*$ is that the numbers $1, \ldots, q_i^*$ have all appeared within the first $i-1$ entries of $\boldsymbol{q}$. Thus, the largest i for which $q_i \leq q_i^*$ is the index where the suffix begins, as q_i must have already appeared prior to index i, and so the value at index i can be increased. Such an index i fails to exist only when we reach the final membership vector $(1, \ldots, n)$.

(ii) Suppose that we selected index i in step (i), and that genotype v_i belongs to episode t. We now have to update the entries at indices $i, i+1, \ldots, M_0 + \ldots + M_t$, which cannot coincide with any earlier entries within episode t. Denote the set of disallowed values as $D = \{q_{M_0+\ldots+M_{t-1}+1}, \ldots, q_{i-1}\}$. We introduce a candidate value c, which is initialised to $q_i + 1$, as the next membership vector we generate must be lexicographically greater. We repeatedly check whether the candidate value c is in D. If $c \in D$, then we increment c by one. Otherwise, we update the entry at index i to c, add c to the set D, increment i by one, and reset c to 1. This process terminates when all entries at indices $i, i+1, \ldots, M_0 + \ldots + M_t$ have all been updated.

(iii) The remaining values of the suffix should take the smallest possible values, which coincide with the initial membership vector. We then store the membership vector, and update the prefix maximum vector using the recurrence relation $q_{i'+1}^* = \max(q_{i'}^*, q_{i'})$ for $i' = i, i+1, \ldots, n-1$.

Putting it together

Enumerating sibling partitions can be done by the function `partitions::setparts` as they are simply the usual set partitions. This can be pre-computed so that we do not need to call the function repeatedly. Note that `partitions::setparts` also returns the partitions in a membership vector format. The function `split` converts the vector format to a partition format (list).

Enumerating identity-by-descent partitions

The function `enumerate_IPs_RG` enumerates all identity-by-descent partitions (IPs) given a relationship graph (RG). The language of clonal and sibling partitions allows the restrictions on identity-by-descent (IBD) partitions to be easily stated: an IBD partition of a relationship graph is a partition over the cells of the clonal partition that must be finer than the sibling partition, and each sibling cell may only contain at most two IBD cells (this is only true because we ignore half siblings).

One way to compute $\mathbb{P}(\boldsymbol{p}|\boldsymbol{g})$ is to enumerate all $\boldsymbol{p}$ compatible with $\boldsymbol{g}$ and consider each equally likely. By definition, all genotypes from the same clonal cell inherit the same allele. For each sibling cell, its clonal cells inherit from one of two parents (two by assumption; a more realistic model would allow for more than two parents, thereby accounting for half-siblings). So the number of IBD partitions over a sibling cell is $2^{\#\text{ clonal cells in sibling cell}-1}$, all of which are equally likely. The -1 term comes from the fact that we divide $2^{\#\text{ clonal cells in sibling cell}}$ by 2 because swapping the two parents results in the same IBD partition. We can apply this argument to each sibling cell to enumerate the IBD partitions over all genotypes (i.e., $2^{\#\text{ clonal cells in first sibling cell}-1} \times 2^{\#\text{ clonal cells in second sibling cell}-1} \times \cdots = 2^{\#\text{ clonal cells} - \#\text{ sibling cells}}$) each having equal probability of $0.5^{\#\text{ clonal cells} - \#\text{ sibling cells}}$.

The function `enumerate_IPs_RG` uses a helper function `split_two` to split each sibling cell of clonal cells into two IBD cells, or to keep the whole sibling cell as an IBD cell. The set of IBD partitions is then the Cartesian product over the splits of each sibling cell. The uniform (log) probability $\mathbb{P}(\boldsymbol{p}|\boldsymbol{g})$ is accounted for by the `log(n.IPs)` term in `RG_inference`.

Equivalently, an obsolete function `compute_pr_IP_RG`, which was superseded by `enumerate_IPs_RG`, computed the probability of an IBD partition given a relationship graph based on a partition model detailed below. Note that both approaches are viable only because we assume the parasite population is entirely outbred. That is to say, we assume parasites have an infinitely large pool of parasite lineages to draw from, such that the probability of drawing the same lineage twice or more from the pool is zero. Strangers draw directly from the pool, thus have

zero probability of being IBD; siblings draw from two parental lineages drawn from the pool, thus are IBD with probability 0.5; clones are copies of a single parental lineage drawn from the pool, thus are IBD with certainty.

$$\mathbb{P}(\boldsymbol{p}|\boldsymbol{g}) = \begin{cases} \text{intra-cell probability if } |\boldsymbol{p}| = 1, \\ \text{intra- and inter-cell probabilities multiplied if } 1 < |\boldsymbol{p}| < n, \\ \text{inter-cell probabilities multiplied if } |\boldsymbol{p}| = n, \end{cases} \tag{1}$$

where

$$\text{inter-cell probability} = \begin{cases} 0 \text{ if there are any clonal edges in } \boldsymbol{g} \text{ that connect cells of } \boldsymbol{p}, \\ 0 \text{ if three or more cells of } \boldsymbol{p} \text{ are interconnected by sibling edges in } \boldsymbol{g}, \\ 0.5^{n_{\text{pairs}}} \text{ otherwise, where } n_{\text{pairs}} \text{ is the number of pairs of cells in } \boldsymbol{p} \\ \text{that are connected by siblings in } \boldsymbol{g}, \end{cases} \tag{2}$$

$$\text{intra-cell probability} = \begin{cases} 0 \text{ if there are any strangers within the cell,} \\ 1 \text{ if there are only clones within the cells,} \\ 0.5^{n_{\text{sibs}}-1} \text{ otherwise, where } n_{\text{sibs}} \text{ is the number of siblings within the cell.} \end{cases} \tag{3}$$

To see the equivalence between the two approaches, note that n_{pairs} is the number of sibling cells that have been split into two IBD cells, which is equal to

$$n_{\text{pairs}} = \#\text{ IBD cells} - \#\text{ sibling cells.}$$

On the other hand, we have for each IBD cell that

$$n_{\text{sibs}} = \#\text{ clonal cells in IBD cell.}$$

Assuming that the IBD partition is consistent with the relationship graph, multiplying the inter-cell probability with all intra-cell probabilities is equivalent to exponentiating 0.5 with

$$\begin{aligned} n_{\text{pairs}} + \sum_{\text{IBD cells}} (n_{\text{sibs}} - 1) &= \#\text{ IBD cells} - \#\text{ sibling cells} + \sum_{\text{IBD cells}} (\#\text{ clonal cells in IBD cell} - 1) \\ &= \#\text{ IBD cells} - \#\text{ sibling cells} + \#\text{ clonal cells} - \#\text{ IBD cells} \\ &= \#\text{ clonal cells} - \#\text{ sibling cells,} \end{aligned}$$

which matches the expression $0.5^{\#\text{ clonal cells}-\#\text{ sibling cells}}$ from above.