

# AN ALGORITHM FOR DISTRIBUTING A RESOURCE AMONG THE VERTICES OF A DYNAMIC DAG

YİĞİT KILIÇOĞLU

**ABSTRACT.** We present an algorithm for distributing a fixed quantity of resources among the vertices of a directed acyclic graph (DAG), where vertices can be toggled on or off, and new vertices and edges may be added over time. Central to the algorithm is a generalization of the genetic coefficient of relationship to arbitrary DAGs.

## 1. PROBLEM STATEMENT

Let  $G = (V, E)$  be a directed acyclic graph (DAG) where  $V$  is the set of vertices and  $E \subseteq V \times V$  is the set of edges. Information about the setup:

- Each vertex  $v$  has an additional property  $\text{on}(v) \in \{0, 1\}$ , which equals 1 if  $v$  is *on*, and 0 if it is *off*.
- The graph is *dynamic* in the sense that new vertices and edges can be added, and the value of a vertex can toggle.
- New edges always originate from an existing vertex and point to a newly added vertex.
- New vertices may have in-degree 0.
- Let  $\alpha$  denote the total amount of resources that need to be distributed among the vertices.

The algorithm should satisfy the following constraints:

- The allocation to vertex  $u$  may decrease due to the addition of a new vertex  $v$  only if there is a path from  $u$  to  $v$  or if  $v$  is an in-neighbor of a vertex that has a path to  $u$ .
- Vertices that are *off* should not be allocated any resources.
- The total allocations across all vertices should sum to  $\alpha$ .

## 2. THE ALGORITHM

Notation:

- $F \subseteq V$ : the set of vertices that have no in-neighbors (the set of *founders*).
- $N^+(v)$ : the out-neighbors of a vertex  $v$ .
- $d^+(v)$ : the outdegree of a vertex  $v$ .
- $N^-(v)$ : the in-neighbors of a vertex  $v$ .
- $d^-(v)$ : the indegree of a vertex  $v$ .
- $D(v)$ : the set of vertices that can be reached from  $v$  via a path of length at least 1 (the set of *direct descendants* of  $v$ ).

The algorithm has two main steps.

**Step 1: Calculating coefficients of relationship.** In genetics, the coefficient of relationship is a measure of genetic relatedness between two individuals [Wri22]. Given parents  $A$  and  $B$  whose coefficients of relationship with a person  $D$  are  $r_{A,D}$  and  $r_{B,D}$ , respectively, the coefficient of relationship between  $D$  and  $C$ , the child of  $A$  and  $B$ , is

$$(1) \quad r_{C,D} = \frac{r_{A,D}}{2} + \frac{r_{B,D}}{2},$$

using the standard definition of the coefficient of relationship. To calculate the coefficient of relationship between a reference individual and everyone else in a family tree, a good shortcut is to apply equation (1) when moving *down* the family tree and halve the coefficient of relationship when moving *up* (at least for the basic cases below). For example, moving from self to parent brings the coefficient of relationship from 1 to 0.5. For intuition, some values are listed below:

Relationship	Coefficient of Relationship
self	1
identical twin	1
parent / child	0.5
full sibling	0.5
grandparent / grandchild	0.25
aunt / uncle / niece / nephew	0.25
half-sibling	0.25
first cousin	0.125
great-grandparent / great-grandchild	0.125
first cousin once removed	0.0625
second cousin	0.03125
first cousin twice removed	0.03125
second cousin once removed	0.0015625

TABLE 1. Coefficients of relationship for common relationships

Observe that family trees are just a special case of DAGs. We will generalize coefficients of relationship to DAGs so that the algorithm, which works on DAGs, can utilize the precision provided by this measure. While each person in a family tree has exactly two parents, vertices in DAGs may have any number of in-neighbors. Our generalization will take this into account.

Let  $r_{v,u}$  denote the coefficient of relationship between vertices  $v, u \in V$  in a DAG. Let  $u_1, u_2, \dots, u_{d^-(v)}$  be the in-neighbors of  $v$ . Let  $r_{u_1,w}, r_{u_2,w}, \dots, r_{u_{d^-(v)},w}$  be their coefficients of relationship with a vertex  $w \in V$ . Define the coefficient of relationship between  $v$  and  $w$  as

$$(2) \quad r_{v,w} := \frac{r_{u_1,w}}{d^-(v)} + \frac{r_{u_2,w}}{d^-(v)} + \dots + \frac{r_{u_{d^-(v)},w}}{d^-(v)}.$$

First, the algorithm sets the coefficients of relationship for the founders. For all  $f \in F$  and for all  $f' \in F \setminus \{f\}$ ,

$$\begin{aligned} r_{f,f} &= 1, \\ r_{f,f'} &= 0. \end{aligned}$$

Then, the algorithm calculates  $r_{v,f}$  for all founders  $f$  and for all vertices  $v$ .

**Step 2: Calculating allocations.** The algorithm starts the allocation with the founders and follows the arrows. One can imagine this as each founder independently distributing their resources to the rest of the DAG. Hence, a vertex's total allocation is the sum of allocations it receives from all founders. For each founder, each vertex  $v$  goes through the following steps:

- (1) Receives some resources from its in-neighbors.
- (2) Takes a portion for itself.
- (3) Distributes the remaining resources among its out-neighbors.

Let  $R_{v,f}$  and  $S_{v,f}$  denote the amount of resources that vertex  $v$  *receives* from its in-neighbors and *sends* to its out-neighbors, respectively, for the process that distributes the resources originating from founder  $f$ . We have

$$(3) \quad R_{v,f} = \sum_{u \in N^-(v)} S_{u,f} \cdot \frac{\frac{1}{d^-(v)} \cdot \max_{x \in D(v) \cup \{v\}} \mathbf{on}(x)}{\sum_{w \in N^+(u)} \frac{1}{d^-(w)} \cdot \max_{x \in D(w) \cup \{w\}} \mathbf{on}(x)}.$$

The max operations ensure that a vertex receives resources from its in-neighbors if and only if either that vertex or at least one of its direct descendants are **on**. Note that the resources a vertex sends to its out-neighbors are not always split equally among them, but instead depend on the number of in-neighbors of each out-neighbor.

Let  $\alpha_{v,f}$  denote the *cut* that vertex  $v$  takes from the resources that it receives from its in-neighbors, for the process that distributes the resources originating from founder  $f$ . No resources should be *lost* in the process, so

$$(4) \quad \alpha_{v,f} = R_{v,f} - S_{v,f}.$$

The amount of resources a vertex takes for itself depends on its direct descendants:

$$(5) \quad \alpha_{v,f} := R_{v,f} \cdot \frac{r_{v,f} \cdot \mathbf{on}(v)}{\sum_{u \in D(v) \cup \{v\}} r_{u,f} \cdot \mathbf{on}(u)}.$$

If  $\sum_{u \in D(v) \cup \{v\}} r_{u,f} \cdot \mathbf{on}(u)$  equals 0 in equation (5), then  $\alpha_{v,f}$  is assigned 0. Because founders distribute resources independently, the total allocation for vertex  $v$  at the end of the algorithm is given by

$$(6) \quad \alpha_v := \sum_{f \in F} \alpha_{v,f}.$$

First, the algorithm sets the amounts *received* by the founders. For all  $f \in F$  and for all  $f' \in F \setminus \{f\}$ ,

$$\begin{aligned} R_{f,f} &= \alpha_f, \\ R_{f,f'} &= 0, \end{aligned}$$

where the  $\alpha_f$  values sum to  $\alpha$ . We leave the choice of  $\alpha_f$  values underspecified, but possible strategies include:

- The founders  $F$  in the initial graph  $G$  get  $\frac{\alpha}{|F|}$  each, and the other founders get 0.
- New founders are allocated the average allocation at the time of their addition to the graph.

The allocations of everyone are adjusted to ensure that they sum to  $\alpha$ .

Finally, the algorithm calculates  $\alpha_v$  for every vertex  $v$  in the DAG.

### 3. ACKNOWLEDGEMENTS

We thank Niranjan Baskaran and Finn Fraser Grathwol for a whiteboard session that accelerated progress on the algorithm, and Ben McDonough and Matan Shtepel for reviewing this paper.

### REFERENCES

- [Wri22] Sewall Wright, *Coefficients of inbreeding and relationship*, The American Naturalist **56** (1922), no. 645, 330–338.

*Email address:* yigit.kilicoglu@aya.yale.edu