MapReduce formulation of PageRank

Maarten Fokkinga

Version of February 11, 2013, 9:19 (Originally: 2010)

For the theoretical discussion, we assume the following be given:

A graph, henceforth called 'the graph'

Variables m and n range over the nodes of the graph

$out\, m \;=\;$ a bag of out links (each one represented by the target node) from node m

$C\, m \;=\;$ the cardinality of $out\, m \;=\; \#out\, m$

PageRank. To simplify the formulas, we use an auxiliary function (for a global constant d):

$$demp\, x \;=\; \frac{1-d}{N} + d \times x$$

Here, N equals the number of nodes in the graph for which a page rank will be computed; some of those nodes might appear in the links but not be given explicitly. Using the *demp* function, the given PageRank specification reads as follows:

$$PR\, m \;=\; demp\,(\frac{PR\, n_1}{C\, n_1} + \cdots + \frac{PR\, n_k}{C\, n_k})$$

where $n_1, \ldots, n_k$ are the nodes that have m in their out links.

Remark. The equation is of the form '$x = \ldots x \ldots$' (where x ranges over functions), or even '$x = f\, x$', when we capture the dots in a function f. We bypass the mathematical theory under what conditions such an equation has a solution for x, and under what conditions the equation has several solutions of which one is the "obviously intended" one. We simply assume that a solution exists and that the desired solution has the form of, and can be computed as, the limit of x_0, $f\, x_0$, $f(f\, x_0)$, $f(f(f\, x_0))$, $\ldots$, for a suitable initial value x_0. We furthermore assume that the limit has been sufficiently approximated when two successive terms in the chain are "near" to each other. This is called "the iterative approach".

Remark. An additional constraint is that $\sum_{m \in nodes} PR\, m$ equals 1; that is, PR is a probability distribution. This constraint is consistent with the equation but doesn't seem to follow from the equation.

Remark. It is allowed that $out\, n$ is a bag rather than a set, that is, node m occurs multiple times in the out links of node n. In that case, n occurs as many times among $n_1, \ldots, n_k$ as the multiplicity of m in $out\, n$. If the corresponding effect on the page rank is undesirable, then one should take care that each $out\, n$ is a set.

Taking a "reasonable" PR_0, we assume that PR is the limit of PR_0, PR_1, PR_2, $\ldots$, where PR_{i+1} is expressed in terms of PR_i according to the page rank specification:

$$(1) \qquad PR_{i+1}\, m \;=\; demp\,(\sum_{n \mid m \in out\, n} \frac{PR_i\, n}{C\, n})$$

Iteration step. The computation of PR_{i+1} from PR_i can be easily formulated as a MapReduce computation once we realize that all PR_{i+1}-values can be computed *simultaneously* from all PR_i-values. To show this by a concrete example, suppose we have nodes a, b, c with outlinks $out\ a = \{a, c\}$, $out\ b = \{c\}$, $out\ c = \{a, b, c\}$. To express PR_{i+1} in terms of PR_i, first rewrite each $PR_i\ m$ as a summation of k times $\frac{PR_i\ m}{k}$, where $k = C\ m$:

$$
\begin{array}{lcccccl}
 & & a & & b & & c \\
\cline{3-7}
PR_i\ a & = & \frac{PR_i\ a}{2} & + & & & \frac{PR_i\ a}{2} & \text{recall: } out\ a = \{a, c\} \text{ and } C\ a = 2 \\
PR_i\ b & = & & & & & \frac{PR_i\ b}{1} & \text{recall: } out\ b = \{c\} \text{ and } C\ b = 1 \\
PR_i\ c & = & \frac{PR_i\ c}{3} & + & \frac{PR_i\ c}{3} & + & \frac{PR_i\ c}{3} & \text{recall: } out\ c = \{a, b, c\} \text{ and } C\ c = 3
\end{array}
$$

Now, a summation and subsequent demping of the columns gives the PR_{i+1}-values:

$$
\begin{array}{lcl}
PR_{i+1}\ a & = & demp\,(\text{“sum of column under } a\text{”}) \\
PR_{i+1}\ b & = & demp\,(\text{“sum of column under } b\text{”}) \\
PR_{i+1}\ c & = & demp\,(\text{“sum of column under } c\text{”})
\end{array}
$$

The partitioning of a's PR-value into the two terms under columns a, c (and similarly for b and c) can be done by a mapper. If, in addition, the column name is added as a key to the terms, then the grouping of the terms per column name is done by the MapReduce framework. Finally, the summation and demping of terms with equal keys (i.e., column names) can be done by a reducer. For our example, the three MapReduce phases read as follows:

The mapper's action:

$$
\begin{array}{lllll}
(a, PR_i\ a) & \xmapsto{mapper} & (a, \frac{PR_i\ a}{C\ a}), & & (c, \frac{PR_i\ a}{C\ a}) \\
(b, PR_i\ b) & \xmapsto{mapper} & & & (c, \frac{PR_i\ b}{C\ b}) \\
(c, PR_i\ c) & \xmapsto{mapper} & (a, \frac{PR_i\ c}{C\ c}), & (b, \frac{PR_i\ c}{C\ c}), & (c, \frac{PR_i\ c}{C\ c})
\end{array}
$$

The grouping of the mapper results gives:

$$
\begin{array}{l}
(a,\ [\frac{PR_i\ a}{C\ a},\ \frac{PR_i\ c}{C\ c}]), \\
(b,\ [\frac{PR_i\ c}{C\ c}]), \\
(c,\ [\frac{PR_i\ a}{C\ a},\ \frac{PR_i\ b}{C\ b},\ \frac{PR_i\ c}{C\ c}])
\end{array}
$$

The reducer's action:

$$
\begin{array}{llll}
(a,\ [\frac{PR_i\ a}{C\ a},\ \frac{PR_i\ c}{C\ c}]) & \xmapsto{reducer} & (a, & demp\,(\frac{PR_i\ a}{C\ a} + \frac{PR_i\ c}{C\ c})) \\
(b,\ [\frac{PR_i\ c}{C\ c}]) & \xmapsto{reducer} & (b, & demp\,(\frac{PR_i\ c}{C\ c})) \\
(c,\ [\frac{PR_i\ a}{C\ a},\ \frac{PR_i\ b}{C\ b},\ \frac{PR_i\ c}{C\ c}]) & \xmapsto{reducer} & (c, & demp\,(\frac{PR_i\ a}{C\ a} + \frac{PR_i\ b}{C\ b} + \frac{PR_i\ c}{C\ c}))
\end{array}
$$

Notice that, when representing of a function by the set of its (input, output)-pairs, the set of inputs for the mapper equals PR_i and the set of outputs from the reducer equals PR_{i+1}. (The above exposition is really convincing, I think, but the appendix gives a *fully formal* proof and even *derivation.*) Thus, representing $out\ m$ by a list, we have:

$$mapper\,(m, p) \quad = \quad [(n, \tfrac{p}{C\ m}) \ \mid\ n \leftarrow out\ m]$$

$$reducer\,(m, ps) \quad = \quad (m,\ demp\,(sum\,ps))$$

and

$$\begin{array}{lll} step & = & mapReduce\ mapper\ reducer \\ PR_{i+1} & = & step\ PR_i \qquad\qquad\qquad \text{-- viewing } PR_j \text{ as a set of (input, output)-pairs} \end{array}$$

Adaptations. One adaptation (and *deviation from the specification*!) is necessary in order to guarantee that PR_{i+1} is a probability distribution again. If $out\,m$ is empty, then the value $PR_i\ m$ vanishes in the iteration step so that the PR_{i+1}-values no longer sum up to 1. For a node with no out links, we may decide that its probability mass is assigned to the node itself (an alternative would be to distribute the probability mass over, say, all nodes of the graph). Thus the adapted mapper reads as follows:

$$mapper\,(m, p) \quad = \quad [(n, \tfrac{p}{C\,m}) \ \mid\ n \leftarrow out\,m] \quad +\!\!+ \quad [(m, p) \ \mid\ out\,m = [\,]]$$

Moreover, another adaptation is needed if the MapReduce computation also has to keep the outgoing links *locally available* for each node so that there is no need for a globally available graph. To achieve this, we assume that each mapper *gets* the pair $(m,\ (p, out\,m))$ instead of just (m, p), and, to make sure that the reducer can construct such pairs again, the mapper *yields* suitable outgoing links in addition to the PR-values:

$$\begin{array}{lll} mapper\,(m,\ (p, ns)) & = & [(n,\ (\tfrac{p}{\#ns},\ \varnothing)) \ \mid\ n \leftarrow ns] \ +\!\!+\ [(m,\ (p, \varnothing)) \ \mid\ ns = [\,]] \ +\!\!+ \\ & & [(m,\ (0,\ ns))] \\ reducer\,(m,\ xs) & = & (m,\ (demp\,(sum\,ps),\ concat\,ns)) \\ & & \text{where } ps = map\,fst\,xs, \qquad ns = map\,snd\,xs \end{array}$$

Iteration. To iterate the step from PR_i to PR_{i+1}, first define when two functions with the same domain are "near" to each other; say, for a given constant *epsilon*:

$$near\,f\,g \quad = \quad max[abs(f\,x - g\,x) \ \mid\ x \leftarrow domain\,f] < epsilon$$

In our case, the PR functions are represented as (input, output)-pairs where each output itself is a (probability, out links)-pair, so that:

$$\begin{array}{lll} near\,pr_1\,pr_2 & = & max[abs(p_1 - p_2) \ \mid\ (n_1, (p_1, ns_1)) \leftarrow pr_1;\ (n_2, (p_2, ns_2)) \leftarrow pr_2;\ n_2{=}n_1] \\ & & < epsilon \end{array}$$

Second, iterate the step until two successive items are near to each other:

$$\begin{array}{lll} iterate\,f\,x_0 & = & x_0\ :\ iterate\,f\,(f\,x_0) \qquad || \quad = [x_0,\ f\,x_0,\ f(f\,x_0),\ f(f(f\,x_0)),\ \ldots] \\ limit\ c\,(x : y : zs) & = & x,\ \text{if } c\,x\,y \\ & = & limit\ c\,(y : zs),\ \text{otherwise} \\ thePR & = & (limit\ near\ .\ iterate\,step)\ initPR \end{array}$$

The definition of *initPR* heavily depends on the way the data is given. Let the graph be given as a collection of $(m,\ out\,m)$-pairs. If we want *initPR* to be a flat probability distribution over all nodes, then we have:

$$initPR \quad = \quad mapReduce\,f\ id\ \text{"the graph"} \qquad\qquad \text{where } f\,(m, ns) = (m, (\tfrac{1}{N}, ns))$$

Appendix

We *formally* prove that "$PR_{i+1} = \mathit{mapeReduce}\ \mathit{mapper}\ \mathit{reducer}\ PR_i$". Actually, we present the proof as a *derivation* of this equation from the defining equation (1). The calculation is concise, machine checkable, and human readable at the same time.

We use '$\cdot$' for function composition. Function application is denoted as usual by a space ' ', having the highest priority, but also by ' . ' having the lowest priority (which is written as '$' in Haskell).

Here is a calculation in which each step tries to manipulate the expression towards the form of an application of *mapReduce*. Along the way we invent a suitable *reducer* and *mapper*. (Of course, this calculation is heavily inspired by our informal exposition above.)

$$
\begin{array}{ll}
 & PR_{i+1} \\
= & \qquad \text{viewing a function as a list of (input, output)-pairs} \\
 & [(m,\ PR_{i+1}\, m) \ \mid\ m \leftarrow \mathit{nodes}] \\
= & \qquad \text{defining equation (1)} \\
 & [(m,\ \mathit{demp}\,(\mathit{sum}\,[\frac{PR_i\, n}{C\, n} \ \mid\ n \leftarrow \mathit{nodes};\ m \in \mathit{out}\, n])) \ \mid\ m \leftarrow \mathit{nodes}] \\
= & \qquad \boxed{\textbf{define}}\ \mathit{reducer}(m, \mathit{xs}) = (m, \mathit{demp}\,(\mathit{sum}\,\mathit{xs})) \\
 & [\mathit{reducer}(m,\ [\frac{PR_i\, n}{C\, n} \ \mid\ n \leftarrow \mathit{nodes};\ m \in \mathit{out}\, n]) \ \mid\ m \leftarrow \mathit{nodes}] \\
= & \qquad \text{map law} \\
 & \mathit{reducer}{*}\ \ .\ \ [(m,\ [\frac{PR_i\, n}{C\, n} \ \mid\ n \leftarrow \mathit{nodes};\ m \in \mathit{out}\, n]) \ \mid\ m \leftarrow \mathit{nodes}] \\
= & \qquad \text{specification } \mathit{groupByKey} \\
 & \mathit{reducer}{*} \cdot \mathit{groupByKey}\ \ .\ \ [(m,\ \frac{PR_i\, n}{C\, n}) \ \mid\ m, n \leftarrow \mathit{nodes};\ m \in \mathit{out}\, n] \\
= & \qquad \text{definition } \mathit{concat} \\
 & \mathit{reducer}{*} \cdot \mathit{groupByKey} \cdot \mathit{concat}\ \ .\ \ [[(m,\ \frac{PR_i\, n}{C\, n}) \ \mid\ m \leftarrow \mathit{nodes};\ m \in \mathit{out}\, n] \ \mid\ n \leftarrow \mathit{nodes}] \\
= & \qquad \text{list law and } \mathit{out}\, n \subseteq \mathit{nodes} \\
 & \mathit{reducer}{*} \cdot \mathit{groupByKey} \cdot \mathit{concat}\ \ .\ \ [[(m,\ \frac{PR_i\, n}{C\, n}) \ \mid\ m \leftarrow \mathit{out}\, n] \ \mid\ n \leftarrow \mathit{nodes}] \\
= & \qquad \boxed{\textbf{define}}\ \mathit{mapper}(n, p) = [(m, \frac{p}{C\, n}) \ \mid\ m \leftarrow \mathit{out}\, n] \\
 & \mathit{reducer}{*} \cdot \mathit{groupByKey} \cdot \mathit{concat}\ \ .\ \ [\mathit{mapper}\,(n,\ PR_i\, n) \ \mid\ n \leftarrow \mathit{nodes}] \\
= & \qquad \text{map law} \\
 & \mathit{reducer}{*} \cdot \mathit{groupByKey} \cdot \mathit{concat} \cdot \mathit{mapper}{*}\ \ .\ \ [(n,\ PR_i\, n) \ \mid\ n \leftarrow \mathit{nodes}] \\
= & \qquad \text{definition } \mathit{mapReduce} \\
 & \mathit{mapReduce}\ \mathit{mapper}\ \mathit{reducer}\ \ .\ \ [(n,\ PR_i\, n) \ \mid\ n \leftarrow \mathit{nodes}] \\
= & \qquad \text{viewing a function as a list of (input, output)-pairs} \\
 & \mathit{mapReduce}\ \mathit{mapper}\ \mathit{reducer}\ \ .\ \ PR_i
\end{array}
$$