In-Rank mesh optimization for URL customized promotion in SEO

Stefan Duprey
Cdiscount
stefan.duprey@cdiscount.com

Second Author
Second University
second.author@university2.com

Abstract

Web site internal mesh optimization is at the very heart of search engine optimization. One prominent way to get the best web search engine visibility for our site is to build the adequate internal linking to promote our naturally popular pages. The definition of popular might be the transformation rate for e-commerce, the traffic from logs for a common site or even a semantic quality rate for our page. We here propose an algorithm to automatically compute the optimal internal mesh for our web site. We tackle the challenges met both at a theory and software implementation level. We'll more specifically deal with big data issues for an e-commerce web site.

Keywords:

search engine, e-commerce, page rank, in rank, mesh optimization, global optimization

I. Introduction

We here propose an original methodology to optimize the internal mesh of Cdiscount e-commerce site. The idea behind is to promote the most successful URLs (the most frequently viewed by users) by increasing their in-rank. The frequency data is obtained from logs parsing tracking software and the in rank is computed using the famous page rank iterative algorithm. We want to find the optimal mesh, which maximizes for all URLs the matching between their traffic and their in-rank. The more successful a URL is, the more in-rank we want to give him through our optimal mesh.

We here detail the technical implementation of such an algorithm. The difficulty here is three-fold:

First the structure of an e-commerce is already well-defined and it is out of question to drastically change the already existing mesh. Links are categorized regarding their incoming page type and position within that page and links can only be created within certain categories. So we'll have strong constraints for our mesh to comply with.

Second the universe we deal with is discrete and gigantic. For a mesh with N nodes, the number of possible meshes is 2^{N^2}. Our objective function is non-linear and non-convex. Exhaustive optimization would be far too computationally intensive. We have to find a proper heuristic based global optimization algorithm to cleverly tweak through our universe.

Third the actual size of our data makes the implementation of a real-world industrial use case a technologically difficult problem. Either we implement from scratch over a cluster our algorithm dealing ourselves with concurrency and inter-processor communications, or we try not to reinvent the wheel and plug ourselves to existing big data platform. The double iterative nature of our algorithm (page rank computations and optimization are both naturally iterative) makes Hadoop Map/Reduce paradigm too slow for our purpose. In contrast to Hadoop's two-stage disk based Map/Reduce paradigm, Spark's in-memory primitives provide the needed performance. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is well suited for our purpose. We'll detail the technological implementation of a real-world use case.

II. Algorithm

Soit $N \in \mathbb{N}$ le nombre de noeuds de notre maillage.

Soit

$$(X_i)_{i\in\{1,\dots,N\}}$$

les sommets de notre graphe orienté.

Soit

$$(G_{ij}) \in \{0,1\}^{N\times N}$$

la matrice adjacente de notre graphe orienté.

$$\max_{(G_{ij})\in\{0,1\}^{N\times N}} \left\{ \sum_{i=1}^{N} trafic\,(X_i) \times PR(X_i) \right\} \tag{1}$$

Initialisation du page rank :

$$\forall u \; PR\,(u) = \frac{1}{N} \tag{2}$$

Calcul itératif :

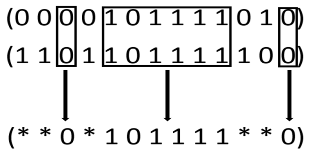
$$PR\,(u) = \frac{(1-c)}{N} + c \times \sum_{v\to u} \frac{PR\,(v)}{card\,(\{v \to u\})} \tag{3}$$

Genetic algorithm is a search heuristic that mimics the process of natural selection A population of individuals to an optimization problem is evolved toward better solutions.

In our case, each individual or chromosome $(C_{ij}) \in \{0,1\}^{N\times N}$ here represented as a bits array, which results from the vectorization of the adjacency matrix (G_{ij})

A candidate can either result from a crossover between two parents or from a self mutation; The crossover function which breeds a child from two parents is defined as follows : The child will keep the matching bits of the two parents and inherit randomly every other non-matching bits. For every two parents, we spawn two children :

Child spawn from 2 parents crossover



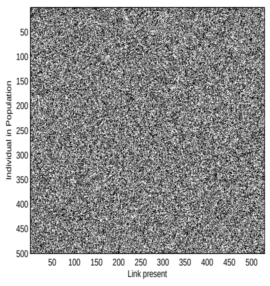
The mutation function is even simpler : we just switch bits whose locations are randomly chosen and proportions equal a fixed mutation rate :

Mutation of an individual

(1 1 0 * 1 0 1 1 * 1 1 0 0)

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation.

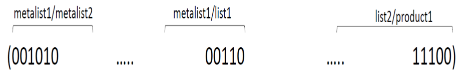
Initial population



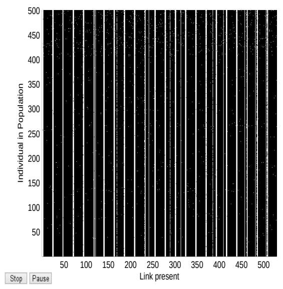
In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Categorization of our links :

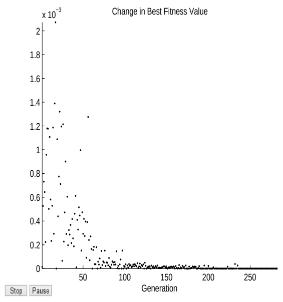
Links categorization



Population at convergence



Best chromosome fitness function evolution



III. Big data implementaino