Heuristic optimization of Distributed Storage Network techniques

Federico Mason*, Davide Peron*, Enrico Lovisotto*
*Department of Information Engineering, University of Padova – Via Gradenigo, 6/b, 35131 Padova, Italy
Email: {masonfed,perondav,lovisott}@dei.unipd.it

***Abstract*—In some previous works about Distributed Storage Network (DSN), two distributed algorithm are presented: Exact Decentralized Fountain Codes (EDFC) and Approximate Decentralized Fountain Codes (ADFC). Unfortunately, the tuning of their fundamental parameter x_d and $\nu(d)$ was not thoroughly investigated.**

We try to solve this problem applying some heuristic optimization techniques to them, in order to obtain the same (or better) results with a proper analysis of each parameter of the problem.

***Index Terms*—Distributed Storage Networks, sensors, heuristic optimization**

I. Introduction

We consider a wide wireless sensor network in which there are lot of sensors distributed over a known region. Some sensors are called *sensing nodes* since they collect data from the environment (temperature, pressure, motion data, ecc..) and send them along the network, the others are called *caching nodes* since they simply store data coming from *sensing nodes*.

To collect data from the network, two approach can be used: centralized or distributed algorithm.

Considering a centralized algorithm, we assume there is a central node that asks for data to each sensing node. This option is the simpler one but it works only in an hypotetical situation in which all the nodes of the network are accessible and the cost of the communication between a central node and the entire network is not prohibitive.

Distributed algorithms are more complex but they works in realistic situation. In a Distributed Storage Network (DSN), when a sensing node has consistent data, it sends them along a random walk between the other nodes. The random walk stops in another node that will store these data.

With these algorithms, a subset of nodes h can be interrogated to retrieve the information of the whole network since in each node are encoded data from one or more sensing nodes of the network. To encode data in each node, *fountain codes* are used.

This approach is firstly presented in [**?**] where they formalize the problem and propose the just described solution given the number of nodes in the network N and the number of sensing nodes K. In their work they don't explain the method used to compute the optimal parameters: x_d for the Exact Decentralized Fountain Codes (EDFC) algorithm and $\nu(d)$ for Approximate Decentralized Fountain Codes (ADFC). Their work is extended in [**?**] removing the assumption of N and K known and they try to estimate them. They also presented some significant plots that estrapolate a relation between parameters of interest. We propose the first version of EDFC and ADFC optimizing respectively x_d and $\nu(d)$ as solution of the problem with some heuristic algorithm and we simulate the distributed algorithm using the previously computed parameters. In details the algorithms we used are Simulated Annealing (SA), Genetic Algorithm (GA) and a modified version of SA that we called Jumping Ball (JB).

The article is structured in 3 sections.

In section II we present in details the heuristic algorithms used, how they work for a general problem and how we adapted them to the specific optimization problem.

In section III we apply SA, GA and JB to the optimization problem and we present the result using the optimal solution in the simulator that we have implemented.

II. Technical Approach

The correct working of the algorithms EDFC and ADFC requires knowledge of x_d and $\nu(d)$. x_d is called *redundancy coefficent* and is the solution of the following optimization problem.

$$\begin{aligned} &minimize \quad \sum_{d=1}^{K} x_d d \mu(d) \\ &subject\ to \quad \begin{cases} Pr(Y < d | X = d) \leq \delta_d \\ x_d \geq 1 \quad for \quad d = 1, ..., K \end{cases} \end{aligned} \tag{1}$$

$\nu(d)$ is a specific distribution and is the solution of the following optimization problem.

$$\begin{aligned} &minimize \quad \sum_{i=1}^{K/R} (\nu'(i) - \mu(i))^2 \\ &subject\ to \quad \begin{cases} \sum_{i=1}^{K} \nu(i) = 1 \\ \nu(i) \geq 0 \quad for \quad i = 1, ..., K \end{cases} \end{aligned} \tag{2}$$

From now we call 1 and 2 simply First Problem (FP) and Second Problem (SP). The different parametrs that appear in FP and in SP are specified in [**?**]. Both FP and SP have *non-convex objective functions* and so it is impossible to determine the exact solution of these problems. To overcome this we build a series of heuristic algorithms.

The techniques we implement doesn't lead to exact solutions but to approximate solutions that are still optimal for our purposes. In particular these techniques are search algorithms that do not stop at the first localized local minimum. They continue to vary the outcome of the objective function, hoping to find a better local minimum than the one previously found. The first algorithm that we have implemented is SA. SA is a probabilistic technique that takes ispiration from annealing in metallurgy, a process that aims to reduce materials defects.

In SA, Objective Function (OF) of the optimization problem is compared to the internal energy of a metallic material. Our algorithm is based on a parameter called Temperature (T). T is initialized at a sufficient high value and then it is reduced until it reaches zero. Every new reduction of T corresponds to a new step of the algorithm. At the first step the algorithm attribuites to the independent variable of OF a random value so that OF respects the constraints of the problem. This value becomes the *candidate* to be the minimum value of OF. Then the *candidate* is perturbed in an unique direction. If the new value found respect the constraints of OF, we keep it. If the new value found doesn't respect the constraints of OF we make a new perturbation. We call the process just described the search of a neighbour of the *candidate*. Once a new admittable value of the OF is found, the algorithm has to establish if it want move to the neighbour of the *candidate* or not. To make this decision SA compares the value of *oc* with the value of *nc*.

In case of New Candidate (NC) has better performance of Old Candidate (OC), that means that OF given by OC is higher than OF given by NC, the algorithm moves to NC for sure. In case that NC has worse performance than OC the algorithm moves to NC only with a certain probability that we call *Acceptance Probability* P_A. This is done so that the algorithm has always the possibility to move away from a local minimum in search of a new solution. The value of the P_A depends on the value of the dofference between the energy of NC and the energy of OC and is given in what follows.

$$P_A = \begin{cases} 1 & F(x_n) - F(x_o) < 0 \\ e^{\psi(T)\cdot(F(x_n)-F(x_o))/T} & F(x_n) - F(x_o) \geq 0 \end{cases} \tag{3}$$

In 3 x_n and x_o represent the value of NC and of OC while $\psi(T)$ is a parameter depending on T called *Acceptance Coefficient*.

For every new value of T the algorithm makes trys to move from the actual candidate a number of times equals to *Steps Coefficient*. We note that *Steps Coefficient* and the perturbation to which the candidates are subjected depend on T as *Acceptance Coefficient*. In particular all these paraeters decreases as the temperature decreases. This means that in initial steps of SA is very easy for the result to change from the starting position while in the final steps is more likely that SA stops at a local minimum.

When T finally reach the zero values SA esmines all the crossed candidates and returns the candidate with better performance. It is therefore essential to always keep in mind the value of the best result going through right now.

```
T = Starting temperature
x = Starting candidate
Step coefficient = Starting step coefficient
Accepatnce coefficient = Starting acceptance coefficient
while(T>0)
```

III. Results

IV. Conclusions And Future Work

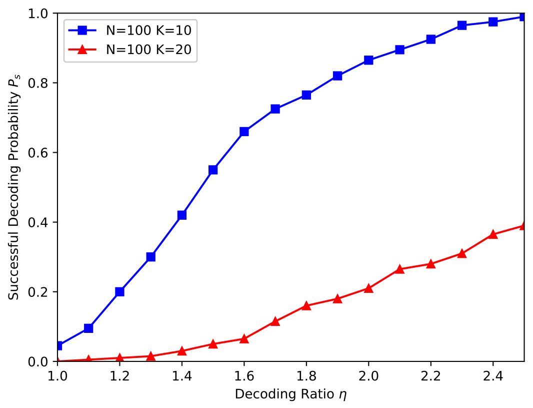

Fig. 1: Successfull decoding probability P_s for different values of N, K and decoding ratio ν.