CentraleSupélec

Algorithmes probabilistes et jeux

Mini-projet: *Online* algorithm

This mini-project will allow you to observe in real life the performance degradation of a deterministic or randomized *online* algorithm compared to one that has the complete information (*offline*). The competitivity ratio will be estimated experimentally; [1]

The validation procedures are:

- Teamwork (in pairs),
- Programming (preferably in `python`; this document contains some tips),
- A public defense: **10 minutes max** the presentation[2] of the approach and results (the presentation material due as a `pdf`) + **5 minutes** of questions (everyone will participate in question sessions) on the morning of Thursday, February 29, 2024,
- A short written discussion (one – two page(s)) of the results obtained and observations made.

The problem to be addressed is that of **Minimum Set Cover** which can be described in its classical version as follows:

Instance:

- a set universe $U = \{0, 1, \ldots, n-1\}$, $|U| = n$,
- a collection $\mathcal{S}$, $|\mathcal{S}| = m$ of subsets of U whose sum covers U: $\mathcal{S} = \{S_0, S_1, \ldots, S_{m-1}\}$, for all $i \in [\![0, m-1]\!]$ $S_i \subseteq U$ and $\bigcup_{S\in\mathcal{S}} S = U$.

Goal: Find a collection $\mathcal{S}'$ such that $\mathcal{S}' \subseteq \mathcal{S}$, $\bigcup_{S\in\mathcal{S}'} S = U$ and $|\mathcal{S}'|$ is **minimum**.

In the *online* version, U and $\mathcal{S}$ are also given. The difference in the formulation of the objective is that the elements to be covered U^* do not necessarily constitute the whole universe, i.e. $U^* \subseteq U$. In addition, the elements of U^* are presented as a sequence $\boldsymbol{\sigma} = (\sigma_i)$, $|\boldsymbol{\sigma}| = \ell$ and for any $i \in [\![0, \ell-1]\!]$ $\sigma_i \in U^*$. Coverage must be assured at all times, i.e. at time i, it must react so that σ_i is covered.

We will award algorithms showing a nice competitive ratio for provided instances.

The tasks to be carried out are thus: design, implementation, and performance evaluation of your *online* algorithms compared to the exact solution.

Although the decision version of the *Minimum Set Cover* problem is *NP*-complete, it is possible to calculate an optimal solution for instances of "reasonable" size. Thanks to the exact solution value provided in the instance file, you can judge the quality of your *online* algorithm by producing the empirical competitiveness ratio for a given instance.

The evaluation criterion is the average of the empirical competitiveness ratios calculated for all the instances supplied in `set_cover_instances.zip` stored on `eCampus`.

The instance format is:

`# Taille univers :` the univers size, $|U| = n$,

`# Taille famille :` the collection size, $|\mathcal{S}| = m$,

`# Famille :` the collection $\mathcal{S}$,

`# Solution exacte :` the exact solution,

[1] We are abusing language here; "our" ratio does not conform to the definition of this measure, which is calculated on all instances.

[2] Useless to recall our project's problem definition; get right to the point!

`# Séquence :` the element sequence, σ.

During the preparatory phase, you can observe the behavior of your deterministic and randomized algorithms. The result of the latter produced for each instance is the average of several launches in order to "smooth" randomness influence.

For the final phase, you choose which of your algorithms seems most promising. If you decide to present a randomized algorithm, you should execute a run at least 30 times and accompany the average computed by the coefficient interval of 95%. To do this, you may inspire yourself with the example below in which `data` gathers a sample of a Gaussian distribution. All you need is to fill it up with the outcomes of your runs.

```
import numpy as np

# find a 95 percent confidence interval for the mean of data
mu, sigma = 0, 1 # mean and standard deviation
data = np.random.normal(mu, sigma, 10000)

confidence_at = 5.
average_value = np.mean(data)
confidence_level =np.percentile(data, [confidence_at/2,100 - confidence_at/2])
print("mean in interval : "+str(average_value)+" inside "+str(confidence_level))
```