The Pagination Problem

Sarah Minich (PhD student) and Pr Imed Kacem (Supervisor)†

†Université de Lorraine
Laboratory of Conception, Optimisation, Modelisation of Systems (LCOMS)

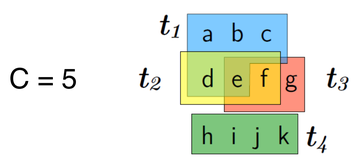
The problem description

The Pagination Problem has four **inputs**.

- A set of n symbols
- A collection of m subsets over these symbols: the tiles
- An infinite number of container in which we are going to store the tiles: the pages
- The integer capacity of each container: C

Goal: find a repartition using as few pages as possible while following two rules:

- All the tiles must be assigned
- All the symbols of a tile must be on a page



Particularity: if two (or more) tiles are on the same page and if they have common symbols, these symbols are not repeated. Thus, it can lead to a great space saving.

Notation: Symbols can have weight that are noted p_i. We denote the sum of all the weights $P = \sum_{i=1}^{n} p_i$

Application

- Symbols → Memory pages
- Tiles → Virtual machines
- Containers → Servers
- The integer capacity

One objective of a provider is to find the optimal scheduling of the Virtual Machines (VM) on his servers in order to earn as much **money** as possible *i.e.* he wants to be able to use as few servers as possible while accessing as many client requests as possible.

Theoretical work

It was proven that the problem is NP-complete which means it is unrealistic to hope to find the optimal value in reasonable time for every possible inputs.
So we took a decision: we agree that we may not have the best solution as long as we are sure that the solution given by our algorithm is not too bad and if we have it in acceptable time.
As we design algorithms **with guaranteed performance**, we have to make sure with mathematical proofs that the distance between the optimal value OPT and the value of the solution returned by our approximation algorithm H is bounded.
The best kind of approximation algorithms we can hope to design are the FPTAS which are described in the next paragraph.

Fully Polynomial Time Approximation Scheme

The FPTAS is a family of algorithms and which is the best technics we have to solve the NP-complete problems. One of the first approximation scheme was proposed in *[3]*. An approximation algorithm is an FPTAS if it meets both following criteria:

- It has a very good time complexity: $poly(\text{input size}, \frac{1}{\epsilon})$
- The distance between the optimal value and the one given by the algorithm is very small: $H \leq (1+\epsilon)OPT$

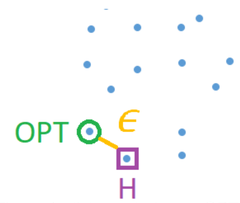


Figure 1: distance between OPT and H

→ The smallest the ϵ, the better the approximation but we pay this precision increase in the time complexity.

One of the special cases

We focused our work for a while on one special case we found in the literature (see [1]). In this special case, we suppose there is a hierarchy in the tiles. This hierarchy can be represented by a tree (see Figure 2).
We proved that this special case is still NP-complete.

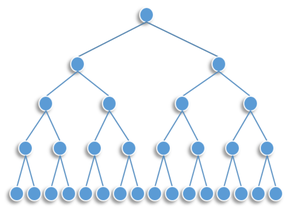


Figure 2: an example of a tree

The main points

We began our work by designing an exact algorithm to solve the special case. We wrote a Dynamic Programming algorithm. Its principle is simple: it will iterate as many times as there are tiles and a during each iteration, the alorithm will process one tile. Thus it will create partial solution at each iteration except during the last one.
The main problem in the DP algorithm is the enormous number of partial solutions it will create during its execution. So we compromised: we will not keep every partial solutions generated. At the end of each iteration, we are going to prune the set of partial solutions and only keep a small number of them. They will be called the representatives.

Theoretical comparison

We designed two FPTAS from the same Dynamic Programming algorithm which is an exact algorithm (for every inputs, it gives the best solution).
Here is a table of the comparison of their theoretical results:

	DP algo	FPTAS 1	FPTAS 2
#Partial sol. generated	$\mathcal{O}(m^2.P)$	$\mathcal{O}(\frac{m^4}{\epsilon^2})$	$\mathcal{O}\left(\frac{m^3}{\epsilon^2}\right)$
Time complexity	$\mathcal{O}(m^3.P)$	$\mathcal{O}(\frac{m^5}{\epsilon^2})$	$\mathcal{O}\left(\frac{m^4}{\epsilon^2}\right)$
Value	OPT	$(1+\epsilon)\ OPT$	$(1+\epsilon)\ OPT$

Computational results

Here are two bar charts we created based on experiments we performed on a laptop computer with a I7 7600U CPU with a frequency of 2.80 GHz and 15.9Go of RAM. We coded our algorithms in Python Anaconda 3.7.1.
We ran our algorithms on inputs with 100 tiles and with an $\epsilon = 0.1$

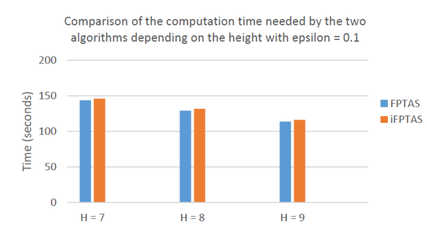


Figure 3: two comparisons

References

[1] Sharing-aware Algorithms for Virtual Machine Colocation, Sindelar, Michael and Sitaraman, Ramesh K. and Shenoy, Prashant, Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2011

[2] Fully Polynomial-Time Approximation Scheme For The Pagination Problem ; Aristide Grange, Imed Kacem, Sébastien Martin, Sarah Minich ; Proceeding of the forty-nineth conference of Computers and Industrial Engineering, Beijing, 18th to 21st of Octobre 2019

[3] Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems, Ibarra, Oscar H. ; Kim, Chul E. ; Journal of the ACM, 10/1/1975, Vol.22(4), pp.463-468; Assciation for Computing Machinery