Hierarchical Clustering of Power Models for circuits design

Adam Desormiere, Lilia Gzara, Jean Bigeon, Luc Nguyen-Thê

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 204 (2022) 566–572

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry Sciences and Computer Science Innovation

Hierarchical Clustering of Power Models for circuits design

Adam Desormiere [a,b,*], Lilia Gzara[a], Jean Bigeon[c], Luc Nguyen-thê [b]

[a] *UDL, INSA Lyon, UCBL, Université Lumière Lyon 2, DISP, EA4570, Villeurbanne, FRANCE*
[b] *Intel France, 166 Rue du Rocher de Lorzier, 38430 Moirans, FRANCE*
[c] *Nantes LS2N UMR CNRS 6004, 2 Chemin de la Houssinière, FRANCE*

Abstract

The work presented in this paper is part of a project which focuses on capitalization and reuse of power models used at Intel to calculate power consumption of electronic devices. These models are analytical and created using an application called IDPA (Intel® Docea™ Power Analytics). Hundreds of thousands of power models have been accumulated in a directory of files and folders, for the simulation of the consumption of thousands of products.
The objective of this work is to group together the models that have been used for the same product, or the same family of products, for example a generation of processors. This notion of project is not present in the current version of the application, and we want to use clustering techniques to make proposals to users wishing to reuse groups of models already present in IDPA database, for example to design the next generation from the current one.
To do that, agglomerative hierarchical clustering is used. Three features are considered to calculate the distance between files in which power models are stored: low delay between files' edition times, similarity of files' names and closeness in filesystem. Hence, we build a tool that can help architects to automatically group their power models into working projects. The proposition made by the algorithm can be refined by an expert or can be directly used by novice users to get an idea on a project on which they have no prior knowledge.

Keywords: Power Modeling, Models Management, Machine Learning, Clustering

* Corresponding author. Tel.: +33-6-69-78-59-82
E-mail address: adam.desormiere@intel.com.

1877-0509
10.1016/j.procs.2022.08.069

1. Introduction

We work with analytical power models that build hierarchically from simple electrical equation models to complex models that simulate the power consumption of real products, such as CPUs. Hundreds of thousands of power models have already been accumulated in our application, for thousands of simulations, and we want to take advantage of this knowledge.

The application Intel® Docea™ Power Analytics (IDPA) [1] is a collaborative power-modeling framework whose objective is to calculate a system's total electrical consumption from its constituent parts and to enable wide exploration of the design space. Hence, that tool helps to discover power-efficient architectures, offering opportunities for productivity gains in power management and time-to-market savings in power/thermal verification and validation.

When we need to model a new generation of processors in IDPA [2], we commonly reuse parts of the previous generation. Currently, we do this manually and store power models in a drive, but this approach can be time-consuming when the files are spread across a storage device. To address that issue, we would like to add to IDPA with the idea of a work project that uses the principles of Product Lifecycle Management (PLM) [3].

As data capitalization and prototype traceability only rely on the knowledge and manual recovery work of experts, PLM could help us to automate it and open it to non-experts [4]. In the current version of IDPA there is no systematic method of labeling or referencing power models to facilitate their reuse for further work of the same type. PLM also enhances, among other things, team collaboration which is a feature that Intel wants to emphasize in its application. For example, by allowing some teams that design power models for high level products (let's say a whole GPU) to use power models of low-level components (like an electrical equation) from another team.

In order to facilitate power models referencing and recovery, we introduce the concept of work project which is a cluster of similar power models. To measure similarities between models, we need to define distances (in part 2). The ideal way to measure similarities between power models is to look at their data (content), but that is time-consuming and resource intensive to open, read, and parse the characteristics of hundreds of thousands of files [5]. In a first step to solving this problem, we only focus the metadata. Among the available metadata we only use the following ones: model name, model location, last editing time [6].

This paper is organized as follows. Section 2 presents clustering approach and techniques that are addressed throughout the paper. Section 3 details our proposed method to calculate similarity between power models. Section 4 presents a case study and results from application of proposed method. Section 5 enumerates the conclusions from our work and introduce future perspectives.

2. Clustering approach

There are many different clustering algorithms [7], including "K Means" [8], "Affinity Propagation" [9], and "Hierarchical Clustering" [10].

K-Means is the most common algorithm. It starts by choosing an integer number (K) of clusters and then assigns each object to the closest cluster, which is calculated at each stage by averaging the values of all the objects in the cluster. The algorithm then recalculates the means for all clusters and assigns each object to the nearest cluster. This process repeats until the clusters stop changing. However, this approach is insufficient since the number of clusters is never known in advance. This is enough to rule out this algorithm.

Another clustering algorithm, Affinity Propagation, makes it possible to find the most representative elements of a set given a "similarity criterion" for the set. This is an iterative algorithm that relies on the sharing of "affinities". For each cluster "c", the algorithm locates a nearby element that is sufficiently similar to it, and then increases its affinity for this element. Subsequent steps propagate the affinity to the other elements. This algorithm performs much better when we can offer a precise description of the objects to be clustered. For example, this approach would be valuable if we compared the data of the power models instead of the metadata (as we do here in our first approach).

Hierarchical clustering is a type of clustering that builds a hierarchy of groups, with each group being a subdivision of a single larger group. This type of clustering is appropriate when the number of clusters is not known *a priori*, since it supports the creation of multiple proposals with different numbers of clusters. Fig. 2 below shows two types of hierarchical clustering, called "Agglomerative" and "Divisive".

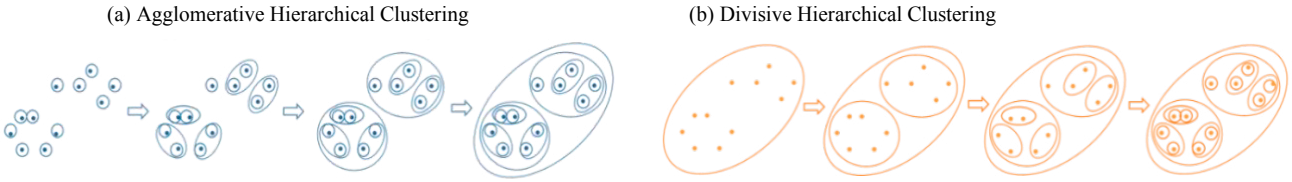

Fig. 1. The two types of Hierarchical clustering: (a) Agglomerative (b) Divisive

Agglomerative clustering is typically faster than Divisive clustering since it creates a hierarchy of clusters in fewer steps. However, Divisive clustering is more flexible since it supports the creation of clusters of any size. We chose to use Agglomerative clustering for our work since time is a serious limitation for such a huge database.

Table 1. Pros and Cons of the clustering algorithms

Clustering algorithms	Pros	Cons
K-means	Scales to large data sets	Request to choose k manually
Affinity propagation	Robust (does not suffer from the initialization), does not request to choose k	Request a precise description of the objects to be clustered
Hierarchical	Does not request to choose k, flexible	Slow, sensitive to noise, computationally demanding

Since the algorithm is stable and convergent [11], we know that after the n^{th} step, we will obtain a single cluster containing all the models. However, we are most interested in the earlier steps n-x, with x growing until we reach a quantity that corresponds to the total number of projects under consideration. Indeed, each of these steps is a clustering proposal for a team's models.

3. Proposed method to define distances between power models

Hierarchical clustering is based on the calculation of distances between the objects to be clustered. These distances depend on the files used for clustering [12]. We define them here using three features to identify the similarities between power models. Hence, we say that two files are "close", and can be clustered, if any of the three following conditions is true:

3.1 The files were last edited in a short time interval

It seems logical that two power circuit models published at about the same time are likely to be part of the same project. Moreover, there is no risk of grouping false positive models between teams working at the same time on different projects because, for privacy reasons, the program is launched only on power models belonging to members of the same business unit. However, we know that within the same team, two (or more) people work in parallel on different projects. We also know projects that have low priority, take a lot of time, or are slightly modified long after initial development (in which case the modifications will not be close in time). This criterion therefore often seems necessary, but neither absolute nor sufficient.

3.2 The files have similar names

We can assume that power models containing similar terms are part of the same project. This is especially true given the technical vocabulary used by power architects. As explained below, a Term Frequency-Inverse Document Frequency (TF-IDF) compares the similarity between the names of power models by vectorizing documents in the space of the total vocabulary (where each dimension is a word), then computing the dot product between each pair of names vectors [13].

The TF-IDF of a term x in a document y is:

$$W_{x,y} = TF_{x,y} * \log \frac{N}{Dfx} \quad (1)$$

with:

- $TF_{x,y}$ is the frequency of the term x in the document y
- Df_x is the number of documents containing the term x
- N is the total number of documents in the corpus

In our case, a document is the name of a model, and the vocabulary is made from all the words of all documents, and length-adjustable n-grams, as proposed in [14].

However, we know that some power models will be used for the design of a product while having very different names (for example, it is plausible that a product needs a model named "RAM" and a model named "CPU", etc... But we hope that these names will be rather "RAM_generation_sku" in order to extract relevant information to associate the models between them. This criterion therefore often seems necessary, but neither absolute nor sufficient.

3.3 The files are close in the filesystem

We need to define the notion of distance between two files in a file and folder tree, which we call filesystem. When we work, the natural tendency is to store files from the same project in the same folder. Similarly, it is plausible that two projects concerning the same product line are stored next to each other in the same parent folder. We count the number of directory changes to go from a model to another. We assume that the closer two files are in the filesystem, the higher the probability that they belong to the same work project. Hence, we find the Lowest Common Ancestor (LCA) for each pair of models, count the number of directory changes needed to go from each model of the pair to the LCA, and sum up the square of the two edges. In the following figure, for example, the black circles are LCAs, the orange circles are models, and the calculation of distance is in black text. At each step, the algorithm calculates these distances for all pairs of models [15].

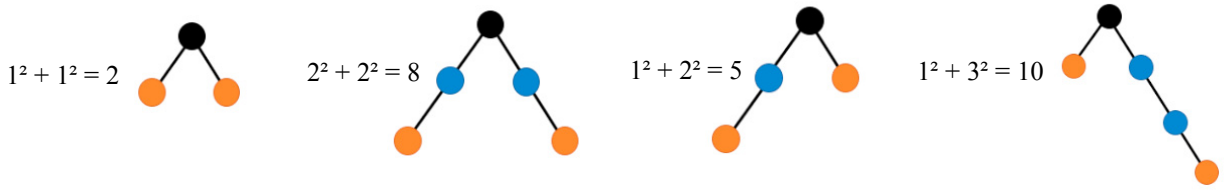

Fig. 2 Example configurations for two power models in the application filesystem.

However, the company and its IDPA application encourage collaboration between teams. For example, a team can create some low-level models, which will then be used by another team to assemble higher-level models. Thus, models from the same project can be stored in different locations in the filesystem, as each team will have stored its models in its own folders. This criterion therefore often seems necessary, but neither absolute nor sufficient.;

4. Application and Results

We store these distances in three square matrices. The coefficient of position (i, j) is the distance between models i and j. These matrices are symmetric, and their diagonal coefficients are all 0 (the distance of a model to itself). We then sum the matrices, weighting with hyperparameters the importance of each feature, before grouping in the same cluster the two closest models and going to the next step. Figure 3 is the distribution of the coefficients for the 3 normalized matrices and the summed matrix. These are well distributed between 0 and 1 and that confirms us in our choice of distances (we expect reasonable results), but could be smoother for the path distance due to the chosen

calculation method.

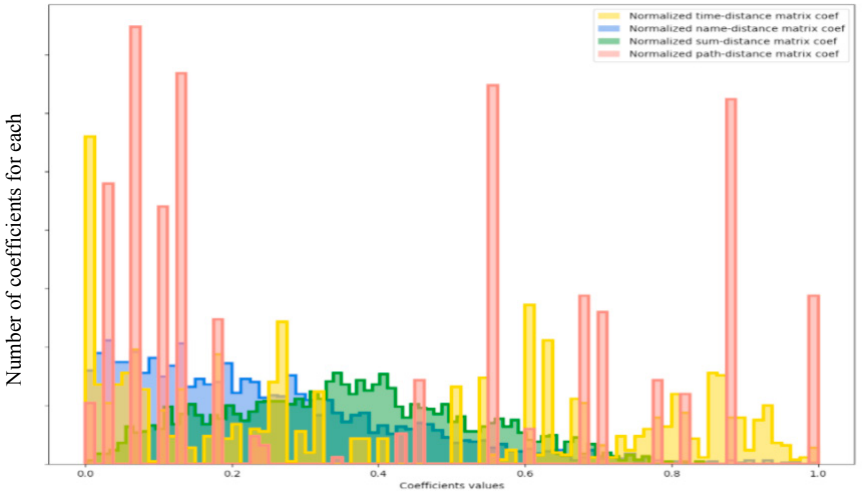

Fig. 3 Distribution of normalized distance matrix coefficients

Summed matrix is defined by the formula:

$$[\text{Sum matrix}] = (\alpha \,.\, [Time\ matrix] + \beta \,.\, [Names\ matrix] + \mu \,.\, [Path\ matrix]) \,.\, \frac{1}{\alpha + \beta + \mu} \quad . \tag{2}$$

We did not undertake a sensitivity analysis, so α, β, μ are equal to 1 by default.

In round 1, there are N clusters and N power models (because each power model forms a cluster on its own). Each round reduces the number of clusters by one (there are N-1 clusters in round 2, for example). After round N, all the power models are in one cluster. In the last rounds and until round N-2, the clustering proposals made by the algorithm were realistic and could be selected by power architects who are very familiar with the database, to arrange their models as working projects. The following dendrogram depicts the clustering process of the algorithm. Reading from bottom to top, we see that when the clusters are grouped together, the distance between them increases.

Figure 4 is a dendrogram that shows this algorithm. It is read from bottom to top. For example, we chose to highlight the N-2 step which shows 3 clusters (the N step has only one cluster left), in 3 different colors. The names of the power models present have been blurred for confidentiality reasons, as they are real models of the application. But we noticed that the models having a common substring, for example "CPU_genX" and "GPU_genX" are directly assembled from the first turns of the algorithm, at the bottom of the figure.

Still for confidentiality reasons, we perform for the moment the clustering within the power models of the same group. We cannot afford to propose to a team a cluster containing models from another team for the moment. We choose for figure 4 to cluster the models of a team with very few models, for readability.

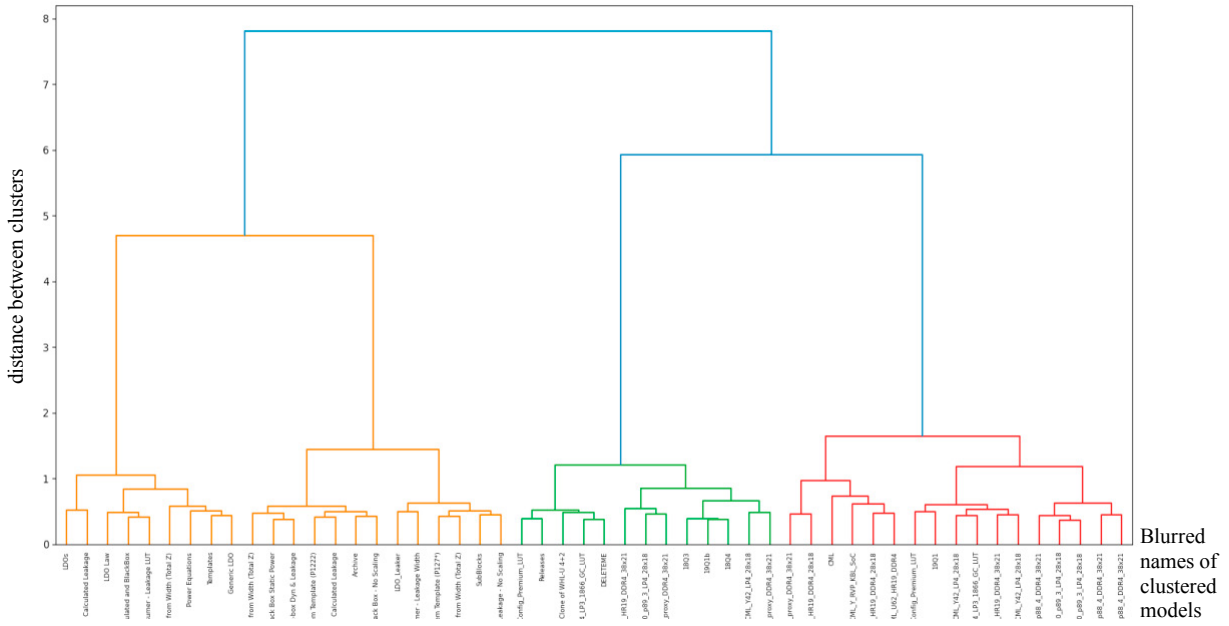

Fig. 4 Dendrogram of Hierarchical Clustering on the models of a Small IDPA Team

5. Conclusion

In conclusion, we have succeeded in proposing a clustering tool that can help architects to automatically group their power models into working projects. Agglomerative hierarchical clustering gathers power models by affinity according to three weighted criteria: low delay between edition times, similarity of files' names and closeness in filesystem.

This proposal can then be refined by the expert, who chooses the stage at which he wants the algorithm to stop, to obtain the clusters and therefore the projects he wants. This can be especially useful for a new user with little or no knowledge of the application's power model database. Of course, this automation does not replace the expert's judgment, but it simplifies the decision making.

As mentioned, one possible improvement to this approach would be to compare the data (content) of the power models rather than the metadata. This requires an in-depth exploration of the constituent elements of the power models, as comparing two models requires the definition of a measure that quantifies the gap between their contents.

Finally, it is not possible to cluster models belonging to different teams for privacy reasons. Thus, our clustering must be performed in a closed space within each team, on the models it owns. We could therefore also consider connecting the cluster boundaries for teams that express the desire to collaborate.

6. Acknowledgments

The authors would like to thank the Association Nationale Recherche et Technologie (ANRT) for its financial contribution. We thank the members of the Docea team from Intel, and members of the Decision & Information Sciences for Production Systems laboratory. We also would like to thank the anonymous paper reviewers who provided good input on the kinds of questions that made sense to address in the final version of the paper.

References

[1] Intel Corporation. (2022) “Intel Docea - Software Trends.” Available Online: https://resources.softwaretrends.com/products/22493/intel-docea.

[2] IEEE Standards Association. (2019) "IEEE Standard for Power Modeling to Enable System Level Analysis."

[3] J. Li, F. Tao, Y. Cheng and L. Zhao. (2015) “Big Data in product lifecycle management.” *International Journal of Advanced Manufacturing Technology* **81 (1-4)**: 667-684.

[4] S. Singh and S. C. Misra. (2018) “Success determinants to Product Lifecycle Management (PLM) performance.” *5th International Conference on Industrial Engineering and Applications - ICIEA 2018,* pp. 386-390.

[5] Y. Shen, Z. Shen, M. Wang, J. Qin, P. H. S. Torr and L. Shao. (2021) “You Never Cluster Alone.” *Advances in Neural Information Processing Systems 34 proceedings*, pp. 1-13.

[6] S. Soltani, S. A. H. Seno and H. S. Yazdi. (2019) “Event reconstruction using temporal pattern of file system modification.” *IET Information Security* **13 (3)**: 201-212.

[7] D. Xu and Y. Tian. (2015) “A Comprehensive Survey of Clustering Algorithms.” *Annals of Data Science* **2 (2)**: 165-193.

[8] P. Li, Y. g. Ding, P. p. Yao, K. m. Xue and C. m. Li. (2016) “Some methods for classification and analysis of multivariate observations.” *Journal of Materials Engineering and Performance* **25 (8)**: 3439-3447.

[9] R. Guan, X. Shi, M. Marchese, C. Yang and Y. Liang. (2011) “Text clustering with Seeds Affinity Propagation.” *IEEE Transactions on Knowledge and Data Engineering* **23 (4)**: 627-637.

[10] F. Nielsen. (2016) "Introduction to HPC with MPI for Data Science." *Springer International Publishing*. ISBN: 9783319219035.

[11] G. Carlsson and F. Mémoli. (2010) “Characterization, stability and convergence of hierarchical clustering methods.” *Journal of Machine Learning Research* **11**: 1425-1470.

[12] M. Oppermann, R. Kincaid and T. Munzner. (2021) “VizCommender: Computing text-based similarity in visualization repositories for content-based recommendations.” *IEEE Transactions on Visualization and Computer Graphics* **27(2)**: 495-505.

[13] P. Bafna, D. Pramod and A. Vaidya. (2016) “Document clustering: TF-IDF approach.” *International Conference on Electrical, Electronics, and Optimization Techniques - ICEEOT 2016,* pp. 61-66.

[14] J. Piskorski and G. Jacquet. (2020) “TF-IDF Character N-grams versus Word Embedding-based Models for Fine-grained Event Classification: A Preliminary Study.” *Proceedings of the Workshop on Automated Extraction of Socio-political Events from News 2020,* pp. 26-34.

[15] W. Xingbo. (2016) “Properties of the Lowest Common Ancestor in a Complete Binary Tree.” *International Journal of Scientific and Innovative Mathematical Research* **3(3)**: 12-17.