HAL
open science

Clustering-based Graph Numbering using Execution Traces for Cache Misses Reduction in Graph Analysis Applications

Régis Audran Mogo Wafo, Thomas Messi Nguélé, Xaviera Youh Djam

▶ **To cite this version:**

Régis Audran Mogo Wafo, Thomas Messi Nguélé, Xaviera Youh Djam. Clustering-based Graph Numbering using Execution Traces for Cache Misses Reduction in Graph Analysis Applications. 2022. hal-03687348v1

HAL Id: hal-03687348

https://hal.science/hal-03687348v1

Preprint submitted on 3 Jun 2022 (v1), last revised 1 May 2024 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Clustering-based Graph Numbering using Execution Traces for Cache Misses Reduction in Graph Analysis Applications

Regis Audran MOGO WAFO[1], **Thomas MESSI NGUELÉ**[1], **Xaviera Youh DJAM** [1]

[1]University of yaounde 1, Department of Computer Science, PO:812, yaounde, Cameroon

*E-mail : {audran.mogo, thomas.messi, xaviera.kimbi}@facsciences-uy1.cm

Abstract

Social graph analysis is generally based on a local exploration of the underlying graph. That is, the analysis of a node of the graph is often done after having analyzed nodes located in its vicinity. However, over the time, networks are bound to grow with the addition of new members, which inevitably leads to the enlargement of the corresponding graphs. At this level we therefore have a problem because more the size of the graph increases, more the execution time of graph analysis applications too. This is due to the very large number of nodes that will need to be treated. Some recent work in-faces this problem by exploiting the properties of social networks such as the community structure to renumber the nodes of the graph in order to reduce cache misses. Reducing cache misses in an application allows to reduce the execution time of this application. In this paper, we argue that combining existing graph ordering with a new numbering that exploit execution traces analysis can allow to improve cache misses reduction and hence execution time reduction. The idea is to build graph numbering using execution traces of graph analysis applications and then combine it with an existing graph numbering (such as cn-order). To build this new ordering, we define a new distance and then used it to analyse execution traces with well known clustering algorithms K-means (for Kmeans-order) and hierarchical clustering (for cl-hier-order). Experiments on a user machine (dual-core) and four cores of Grid'5000 node (Neowise) show that this combination improves slightly existing graph ordering (cn-order, numbaco, rabbit and gorder) in almost all the cases (the two cores of dual-core, all the four cores of neowise), with PageRank graph application and astro-ph dataset. For example, on neowise with one thread and Astro-ph dataset, the best performance is given with the combination kmeans-order_cn-order which allows to reduce by 42.59% the cache misses (compared to the second numbaco with 40.79%) and therefore by 7.27 % the time of execution (compared to 6.89% for the second numbaco).

Keywords

Graph analysis; Execution trace; Machine learning

I INTRODUCTION

Actors and their interconnections in social networks are modeled with graphs where nodes are actors and links are their interconnections. The advent of computers and communication networks allows to analyze data (see on [2]) on a large scale and has lead to a shift from the study of individual properties of nodes in small specific graphs with tens or hundreds of nodes, to the analysis of macroscopic and statistical properties of large graphs also called complex networks, consisting of millions and even billions of nodes [2].

Social networks welcome new members every day. This contributes to enlarging the size of the corresponding graphs. It should be noted that the larger the graph, the longer the execution of the analysis applications takes time [5]. This usually results from a large number of [7] cache misses, themselves caused by many memory accesses in search of certain nodes during the execution of the graph analysis application.

Graph application consists in analyzing various nodes; process a node usually involves the analysis of the other ones located in its vicinity. Reducing cache misses in an application allows to reduce the execution time of this application. In order to reduce cache misses, the main idea is therefore to renumber the graph such away that the nodes likely to be processed together become close in the memory. Recent numbering algorithms guided by this idea have been proposed such as Cn-order[8–10], Gorder [6], Rabbit order [5] and NumBaCo [7]. They stood out for their efficiency compared to other existing algorithms (see in [8–10]). In this paper we use the execution traces analysis to build these new numbering, and we combine them with one of the recent numbering algorithms in order to have better results.

The remainder of this paper is structured as follows. Section II presents recent works on graph ordering. Section III introduces our main contribution. Experimental results are shown in section IV. Section V is devoted to the synthesis of our contribution, together with the conclusion and future work.

II RELATED WORK

For several years, graph reordering algorithms have attracted the attention of researchers and this attention is constantly increasing. The most recent graph reordering include NumBaCo Messi Nguélé, Tchuente, and Méhaut [7], Gorder Wei, Yu, Lu, and Lin [6], Rabbit Order Arai, Shiokawa, Yamamuro, Onizuka, and Iwamura [5] and Cn-order Nguélé, Tchuente, and Méhaut [8]. Table1, gives a summarize of the existing numbering methods with their advantages and their limits.

The main idea of **NumbaCo** is to exploit the community structure of graph nodes to improve graph analysis applications performances through cache misses reduction (see [7]).

In the case of **G-order**, Wei, Yu, Lu, and Lin [6]) offered an order that allows nodes in the direct neighborhood to be close in memory.

Rabbit-order proposed by Arai, Shiokawa, Yamamuro, Onizuka, and Iwamura [5]) proposes a numbering method based on two approaches namely:

- – Scheduling based on the community structure of the graphs in the real world: they try to match the hierarchical communities of the graphs in the real world with the hierarchies located at the level of the caches of the central processor unit;
- – An incremental aggregation in parallel: this one aggregates in an incremental way the vertices of the same community in parallel, which has the consequence of reducing the number of vertices to be processed.

Cn-order proposed by Nguélé, Tchuente, and Méhaut [8] is based on a fusion of the advantages of previous models such as:

- grouping in memory nodes appearing frequently in direct neighborhood (based on G-order)
- grouping in memory nodes belonging to the same community or sub-community (based on Numbaco and Rabbit-order).

None of these methods doesn't exploit former executions of target application. In this paper, we argue that exploiting former execution of the target applications (for example, exploiting execution traces analysis) can allow to improve existing performances.

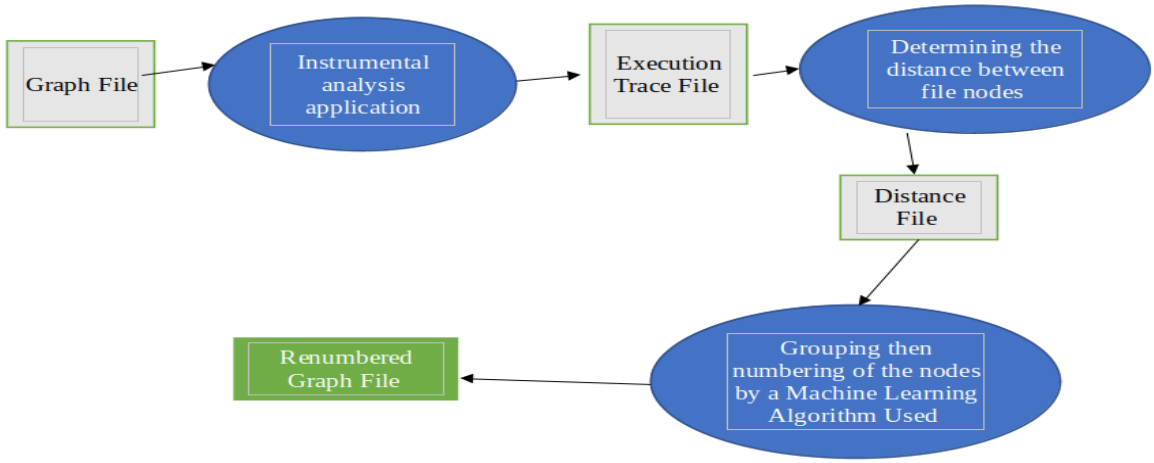
Table 1: Comparison of existing numbering methods

Reference	Year	Techniques	Specifications	Limits
[7]	2015	NumBaCo	Exploits community structure of the graph in order to renumber it	Doesn't take into account nodes degrees heterogeneity Doesn't exploit former executions of target applications
[6]	2016	Gorder	Consider nodes sibling nodes	Does not take into account communities Doesn't exploit former executions of target applications
[5]	2016	Rabbit-order	Based on community structure + is parallel	Doesn't taking into account nodes degrees heterogeneity Doesn't exploit former executions of target applications
[8]	2017	Cn-order	Combines advantages of the others	Doesn't exploit former executions of target applications

III NUMBERING BASED ON EXECUTION TRACES ANALYSIS

The idea here is to build a graph numbering by analyzing execution traces of graph applications with clustering algorithms. In this section, we first present execution traces in our case (section 3.1), then we present a distance (see section 3.2) that allows us to build our new numbering algorithm (see section 3.3). Figure 1 present the complete principle of numbering using execution traces analyses.

Figure 1: Principle of numbering by execution traces analyses



3.1 Execution traces

The execution traces allow to see the behavior of the application during its execution. Since it is a graph analysis application, we are interested to keep the way nodes are accessed during execution. To obtain these traces, we add in the graph analysis program some code that allow to print at each line of a file, a node and the list of other nodes directly encountered during the execution program.

The goal of our work is to analyze this execution traces file in other to build a numbering that will allow "closer nodes" to have "closer numbering". To carry out this work, we choose to analyze these execution traces with clustering algorithms such as Kmeans and Hierarchical clustering. Those algorithms use with a distance that measures the closeness between nodes of the execution traces file.

The next section present the approach used to compute the distance between nodes in execution traces file.

3.2 Distance between two nodes

Alex Groce [3] defines the distance between two execution traces x and y of the same program as follows: $\bar{D}(x,y) = \sum_{i=0}^{n} \Delta(i)$, $where\ \Delta(i) = \begin{cases} 0,\ if\ \ val_i^x = val_i^y \\ 1,\ if\ val_i^x \neq val_i^y \end{cases}$

val_i^x and val_i^y are the value of x and y at the time i.

Inspired by this, we define a metric between two nodes a and b encountered in the execution traces file.

Proximity D between two nodes. Let $\Gamma(a)$ and $\Gamma(b)$ the set of nodes encountered directly after a and b respectively. The proximity D between a and b is defined as follows:

$$D(a,b) = |\Gamma(a) \cup \Gamma(b)| - |\Gamma(a) \cap \Gamma(b)|$$
$$= (|\Gamma(a)| + |\Gamma(b)| - |\Gamma(a) \cap \Gamma(b)|) - |\Gamma(a) \cap \Gamma(b)|$$
$$= |\Gamma(a)| + |\Gamma(b)| - 2|\Gamma(a) \cap \Gamma(b)|$$

***Theorem** III.1: D is a distance*
The proximity D between two nodes as defined is a distance.

The complete proof of theorem is on the Annex (see in A).

3.3 Proposed Algorithms

In this section, we present numbering based on execution trace analysis and the combinations between these algorithms and the previous existing ones.

3.3.1 Kmeans-order and Cl-hier-order

Algorithm 1 : **Kmeans-order**

Input: $G = (V, E), K, File, D,$
Output: G''

1: $G'' \leftarrow Kmeans(K, G, File, D)${ K is the desired number of cluster, G input graph, File contain the execution traces, D distance defined in section3.2}
2: Return G''

Algorithm 2 presents clustering based on hierarchical clustering.

Algorithm 2 : **Cl-hier-order**

Input: $G = (V, E), File, D$
Output: G''

1: $G'' \leftarrow HierClus(G, File, D)${ G input graph, File contain the execution traces, D distance defined in section3.2}
2: Return G''

3.3.2 Combination between Cn-order with Kmeans-order and Cl-hier-order

Algorithm 3 present the combination kmeans-order_cn-order.

Algorithm 3 : **Kmeans-order_cn-order**

Input: $G = (V, E), K, File, D$
Output: G''

1: $G' \leftarrow Kmeans-order(K, G, File, D)${ K is the desired number of cluster, G input graph, File contain the execution traces, D distance defined in section3.2}
2: $G'' \leftarrow existing - numbering(G')${ Renumbering graph G' resulting of kmeans-order }
3: Return G''

Algorithm 4 present the combination Cl-hier-order_cn-order.

Algorithm 4 : **Cl-hier-order_cn-order**

Input: $G = (V, E), File, D$
Output: G''

1: $G' \leftarrow HierClus(G, File, D)${ G input graph, File contain the execution traces, D distance defined in section3.2}
2: $G'' \leftarrow existing - numbering(G')${use cn-order for Renumbering graph G' resulting of Cl_hier-order }
3: Return G''

Algorithm 5 present the combination Cn-order_kmeans-order.

Algorithm 5 : **Cn-order_kmeans-order**

Input: $G = (V, E), K, File, D$
Output: G''

1: $G' \leftarrow existing - numbering(G)$ {use cn-order for numbering graph G }
2: $G'' \leftarrow Kmeans(K, G, File, D)$ { using Kmeans-order for renumbering G' }
3: Return G''

Algorithm 6 present the combination Cn-order_Cl-hier-order.

Algorithm 6 : **Cn-order_Cl-hier-order**

Input: $G = (V, E), File, D$
Output: G''

1: $G' \leftarrow existing - numbering(G)$ {use cn-order for numbering graph G }
2: $G'' \leftarrow HierClus(G, File, D)$ { using cl_hier-order for renumbering G' }
3: Return G''

IV EXPERIMENTAL EVALUATION

The experiments were done two platforms:

- A user machine with the following characteristics: 2 cores at 1.80GHz, 4 GB Ram, L3 of 2048 KB, L2 of 256 KB and L1 of 32KB.
- A node of Grid'5000 platform (Neowise) with these characteristics: 10 nodes, each node has 1 CPU AMD EPYC 7642, each CPU has 48 cores, 512GB RAM, L3 of 256 MB.

For this evaluation, we present results got with the well known graph analysis application, Pagerank [1]. We used a Posix thread implementation proposed by *Nikos Katirtzis*[1]. This implementation uses adjacency list representation. We use Astro-ph dataset [4] that has n=16,046 nodes and m=242,502 edges (that is almost **2.46 MB** with the formula presented at Messi Nguélé and Méhaut [10], the graph size is $G_space = 8m + 40N\ bytes$).

In this section, we compare previously proposed graph numerations (cn-order, rabbit, gorder, numbaco) with the numeration induced by execution traces analysis in three ways:cache reference reduction (section 4.1), cache-misses reduction (section 4.2), execution time reduction (section 4.3). We do this task with two tables:

- Table 2 that compares cache references, cache misses and execution time on the user machine described above;
- Table 3 that does the same but on four cores of Grid'5000 node (Neowise).

Color signification is as follows: colored numbers on a column express the four best, the two first are bold (blue for the time, red for the cache misses and orange for the cache references).

[1] https://github.com/nikos912000/parallel-pagerank

Table 2: Graph Ordering Comparison (Pagerank, Astro-ph, dual-core machine)

Heuristic	**Time** (ms)	**Cache misses**	**Cache references**
Astro-ph 1th	607.29475	2832481.5	16569884.75
Gorder 1th	570.486 (6.06%)	2389119.25 (15.65%)	9563592.75 (42.28%)
NumBaCo 1th	571.9995 (5.81%)	2423346.5 (14.44%)	8487259.75 (48.78%)
Rabbit 1th	575.08125 (5.30%)	2536938.75 (10.43%)	9233646.25 (44.27%)
Cn-o 1th	**566.20725 (6.76%)**	**2270193.5 (19.85%)**	8237936 (50.28%)
k-means-o 1th	618.528 (-1.85%)	3044132.75 (-7.47%)	17330640.75 (-4.59%)
cl-h-o 1th	594.1935 (2.16%)	2522649.5 (10.94%)	15282698.25 (7.77%)
Cn-o_k-means-o 1th	572.231(5.77%)	2398583.25 (15.32%)	8488329.75 (48.77%)
Cn-o_cl-h-o 1 th	585.329 (3.62%)	2420409.25 (14.55%)	14831642.25 (10.49%)
k-means-o_Cn-o 1 th	**564.867 (6.99%)**	**2248500 (20.62%)**	**8193249.75 (50.55%)**
cl-h-o_Cn-o 1th	569.471 (6.23%)	2357463.75 (16.77%)	**8172304.5 (50.68%)**
Astro-ph 2th	475.46975	3455467	18489284.5
Gorder 2th	536.18175(-12.76%)	2964778.75(14.20%)	11210821 (39.36%)
NumBaCo 2th	419.18575(11.83%)	2958648.25(14.37%)	10173745.5 (44.97%)
Rabbit 2th	**402.621(18.09%)**	3026377 (12.41%)	11077611.25 (40.08%)
Cn-o 2th	**408.30775(14.12%)**	**2761599(20.08%)**	**9923092 (46.33%)**
k-means-o 2th	520.92475(-9.56%)	3739659 (-8.22%)	19466981 (-5.28%)
cl-h-o 2th	452.037(4.92%)	3007137.25(12.97%)	17509240.5 (5.30)
Cn-o_k-means-o 2th	460.95275(3.05%)	2950742(14.60%)	10542184.25 (42.98%)
Cn-o_cl-h-o 2 th	432.43575 (9.05%)	3000986(13.15%)	16928641 (8.44%)
k-means-o_Cn-o 2 th	437.171 (8.05%)	2833740.5 (17.99%)	9975563.25 (46.04%)
cl-h-o_Cn-o 2th	449.738(5.41%)	**2819274.75(18.41%)**	**9858212.75 (46.68%)**

Table 3: Graph Ordering Comparison (Pagerank, Astro-ph, 4 thread, Neowise[grid'5000])

Heuristic	Time (ms)	Cache misses	Cache references
Astro-ph 1th	266.905	10276724.75	42293956
Gorder 1th	250.212 (6.25%)	6761279 (34.20%)	27140765.5 (35.82%)
NumBaCo 1th	**248.50975 (6.89%)**	6084717.75 (40.79%)	29021769.5(31.38%)
Rabbit 1th	253.89025(4.87%)	7325485.25(34.57%)	27670585(34.57%)
Cn-o 1th	249.2455 (6.61%)	6220675 (39.46%)	**26241990 (37.95%))**
k-means-o 1th	269.25375(-0.87%)	10216968(0.58%)	39365789.5(6.92%)
cl-h-o 1th	256.76975 (3.79%)	7417636(27.82%)	54311434(-28.41%)
Cn-o_k-means-o 1th	250.4(6.18%)	6437387.75(37.35%)	**25798704.75 (39.00%)**
Cn-o_cl-h-o 1 th	255.4765(4.28%)	7529694.25(26.73%)	48205558.75(-13.97%)
k-means-o_Cn-o 1 th	**247.4955 (7.27%)**	**5899736.5 (42.59%)**	27898282 (34.03%)
cl-h-o_Cn-o 1th	249.75525 (6.42%)	**5818206.25 (43.38%)**	27789259 (34.29%)
Astro-ph 2th	184.07075	11069265	48044309.75
Gorder 2th	209.2585(-13.68%)	7236403(34.62%)	32165345 (33.05%)
NumBaCo 2th	158.03575(14.14%)	6797367.75(38.59%)	33022849.75(31.26%)
Rabbit 2th	148.01475 (19.57%)	7976793.25(27.93%)	35010926.75 (27.12%)
Cn-o 2th	164.0755(10.86%)	6765081(38.88%)	31541515 (34.35%)
k-means-o 2th	184.31575(-0.13%)	11660819(-5.34%)	46860805.5 (2.46%)
cl-h-o 2th	**143.2345 (22.18%)**	9089591.5(17.88%)	61258435 (-27.50)
Cn-o_k-means-o 2th	165.903(9.86%)	7194631.5(35.00%)	**30722091.25 (36.05%)**
Cn-o_cl-h-o 2 th	**139.3905 (24.27%)**	8760113.75(20.86%)	56590441.75 (-17.79%)
k-means-o_Cn-o 2 th	162.779(11.56%)	**6751559(39.00%)**	**31312716.75 (34.82%)**
cl-h-o_Cn-o 2th	157.9145 (14.20%)	**6754098.25(38.98%)**	31498630.5 (34.43%)
Astro-ph 3th	136.47175	12759681.25	54466648.25
Gorder 3th	173.27525(-26.23%)	7975692.5(37.49%)	35949448.5 (33.99%)
NumBaCo 3th	**120.644(11.59%)**	7749122.5(39.26%)	36747591.5 (32.53%)
Rabbit 3th	**110.48925(19.03%)**	9079863.5(28.83%)	40484777.75 (25.67%)
Cn-o 3th	124.16175(9.02%)	**7613301.75(40.33%)**	34903471.25 (35.91%)
k-means-o 3th	137.493(-0.74%)	12949269.75(-1.48%)	52550052.75 (3.51%)
cl-h-o 3th	135.2245(0.91%)	9967014.5(21.88%)	61710753.75 (-13.30%)
Cn-o_k-means-o 3th	125.599(7.96%)	8101565.5(36.50%)	34615031.75 (36.45%)
Cn-o_cl-h-o 3 th	133.95425(1.84%)	9917300.75(22.27%)	57836840.25 (6.18%)
k-means-o_Cn-o 3 th	129.6565(4.99%)	7724722(39.45%)	**34273040.25 (37.07%)**
cl-h-o_Cn-o 3th	121.89275(10.68%)	**7493254(41.27%)**	**34177652.75 (37.25%)**
Astro-ph 4th	266.07925	12270609.5	54432887.75
Gorder 4th	245.762(7.63%)	7763059.75 (36.73%)	38749439 (28.81%)
NumBaCo 4th	**233.7525(12.14%)**	7508885.25 (38.80%)	38274430.75 (29.68%)
Rabbit 4th	247.38(7.02%)	8379379.5 (31.71%)	40160100.5 (26.22%)
Cn-o 4th	**231.7295(12.90%)**	7408551.5 (39.62%)	**36283533.5 (33.34%)**
k-means-o 4th	289.7065(-8.87%)	11881925.25 (3.16%)	52188152.5 (4.12%)
cl-h-o 4th	305.36675(-14.76%)	9241197.75 (24.69%)	64502363.25 (18.49%)
Cn-o_k-means-o 4th	251.41425(5.51%)	7714594.25 (37.13%)	36841011.75 (32.31%)
Cn-o_cl-h-o 4 th	298.334(-12.12%)	8885607.5 (27.59%)	62234059 (-14.33%)
k-means-o_Cn-o 4 th	242.103(9.01%)	**7386400.25 (39.80%)**	36492023.25 (32.95%)
cl-h-o_Cn-o 4th	242.649(8.80%)	**7394821.25 (39.73%)**	**36160439.75 (33.56%)**

4.1 Cache-References Reduction

Cache references correspond to the number of time the program (Pagerank in this case) looks for data at L3 cache memory. So when the data required by the processor is not present at L1 and L2 cache memory, it is looked at L3 cache memory and this causes a cache reference. This means that, the number of cache references increases with the increasing of L1 and L2 cache misses.

In Table 2, we can see that:

- With one thread, the two combinations (Kmean-order_cn-order) and (cl-hier-order_cn-order) produce the best cache-references reduction 50.55% and 50.68% respectively. Cn-order is the third with 50.28%.

- With two threads, the combination (cl-hier-order_cn-order) produces the best reduction with 46.68 %, Cn-order is the second with 46.33% and the combination (Kmean-order_cn-order) is the third with 46.04%.

In Table 3, we can remark that:

- With one thread, the combination (cn-order_Kmean-order) produces the best cache-references reduction 39%, Cn-order is the second with 37.95% and Gorder is the third with 35.82%.
- With two threads, the combinations (cn-order_Kmean-order, Kmean-order_cn-order, cl-hier-order_cn-order) produce the best reduction with 36.05%, 34.82% and 34.43% respectively.
- With three threads, the combinations (cl-hier-order_cn-order, Kmean-order_cn-order, cn-order_Kmean-order) produce the best reduction with 37.25%, 37.07% and 36.45% respectively.
- With four threads, the combination (cl-hier-order_cn-order) produces the best reduction with 33.56%, Cn-order is the second with 33.34% and the combination (Kmean-order_cn-order) is the third with 32.95%.

According to these observations, we can say that the combination of cn-order with the numbering induces by execution traces analysis (kmeans-order and cl-hier-order) allows to reduce cache references more than existing graph ordering heuristics. Therefore, using execution traces analysis allows to reduce cache misses on L1 and L2 cache memory.

4.2 Cache-Misses Reduction

Cache misses correspond to the number of time the program looks for data at L3 cache memory and doesn't find it. When the data required by the processor is not present at L3 cache memory, it is looked at central memory and this causes a cache miss.

In Table 2, we can see that:

- With one thread, the combination (Kmean-order_cn-order) produces the best cache-misses reduction 20.62%, Cn-order is the second with 19.85% and the combination (cl-hier-order_cn-order) is the third with 16.77%.
- With two threads, cn-order is the first with 20.08%, the combination (cl-hier-order_cn-order) is the second with 18.41% and the combination (Kmean-order_cn-order) is the third with 17.99%.

In Table 3, we remark that:

- With one thread, the combinations (cl-hier-order_cn-order) and (Kmean-order_cn-order) produces the best cache-misses reduction with respectively 43.38% and 42.59%. NumBaCo is the third with 40.79%.
- With two threads, the combinations (Kmean-order_cn-order, cl-hier-order_cn-order) produce the best reduction with 39.00% and 38.98% respectively, and cn-order is the third with 38.88% .
- With three threads, the combination (cl-hier-order_cn-order) produces the best reduction with 41.27%, cn-order is second with 40.33% and the third is the combination (Kmean-order_cn-order,) with 39.45% .
- With four threads, the combinations (Kmean-order_cn-order , cl-hier-order_cn-order) produces the best reduction with 39.80% and 39.73% respectively. NumBaCo is the third with 38.80%.

These observations show that the combination of cn-order with the numbering induces by execution traces analysis (kmeans-order and cl-hier-order) is almost always among the best in terms of cache misses reduction compare to existing graph ordering heuristics. Therefore, using execution traces analysis allows to reduce cache misses on L3 cache memory.

4.3 Execution Time Reduction

Execution time correspond to duration taken by the program (Pagerank in this case) to analyse the graph passed to it as parameter.

In Table 2, we can see that:

- With one thread, the combination (Kmean-order_cn-order) produces the best time reduction 6.99%, cn-order is the second with 6.76% and the combination (cl-hier-order_cn-order) is third with 6.23%.
- With two threads, Rabbit, cn-order and NumBaCo produces the best reduction with 18.09%, 14.12% and 11.83% respectively. The combination (cn-order_cl-hier-order) is fourth with 9.05%.

In Table 3, we can remark that:

- With one thread, the combination (Kmean-order_cn-order) produces the best time reduction with 7.27%, NumBaCo is the second with 6.89% and cn-order is the third with 6.61%.
- With two threads, the combination (cn-order_cl-hier-order) produces the best reduction with 24.27%, cl-hier-order is second with 22.18%, Rabbit-order is the third with 19.57%.
- With three threads, Rabbit-order produces the best reduction with 19.03%, NumBaCo is second with 11.59% and the combination (cl-hier-order_cn-order) is third with 10.68%.
- With four threads, Cn-order produces the best reduction with 12.90%, NumBaCo is the second with 12.14% and the combination (Kmean-order_cn-order) is the third with 9.01%.

These observation show that, even with the execution time reduction, the combination of cn-order with the numbering induces by execution traces analysis (kmeans-order and cl-hier-order) is almost always among the best compare to the existing graph ordering heuristics.

V CONCLUSION

In this paper, we proposed a new numbering that exploit execution traces analysis in order to improve cache misses reduction and hence execution time reduction in graph application. To build this ordering, we defined a new distance and then we used it to analyse execution traces with well known clustering algorithms K-means (for Kmeans-order) and hierarchical clustering (for cl-hier-order). This numbering is intended to be combined with existing graph numbering (such as cn-order). We showed on a user machine (dual-core) and four cores of Grid'5000 node (Neowise) that the gotten combination improves slightly existing graph ordering (cn-order, numbaco, rabbit and gorder) in almost all the cases with PageRank and astro-ph dataset. For example, on neowise with one thread, the best performance is given with the combination kmeans-order_cn-order which allows to reduce by 42.59% the cache misses (compared to the second numbaco with 40.79%) and therefore by 7.27 % the time of execution (compared to 6.89% for the second numbaco).

During the executions, we saw that the new numbering takes long execution time due the distance computation. So, build this numbering could be unfeasible with huge graphs. Therefore a

direct perspective could be to see the way to reduce the time taken by the distance computation during clustering.

Publications

[1] L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank citation ranking: bringing order to the web." In: (1999).

[2] M. E. Newman. "The structure and function of complex networks". In: *SIAM* 45.2 (2003), pages 167–256.

[3] A. Groce. "Error explanation with distance metrics". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2004, pages 108–122.

[4] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[5] J. Arai, H. Shiokawa, T. Yamamuro, M. Onizuka, and S. Iwamura. "Rabbit order: Just-in-time parallel reordering for fast graph analysis". In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2016, pages 22–31.

[6] H. Wei, J. X. Yu, C. Lu, and X. Lin. "Speedup graph processing by graph ordering". In: *Proceedings of the 2016 International Conference on Management of Data*. 2016, pages 1813–1828.

[7] T. Messi Nguélé, M. Tchuente, and J.-F. Méhaut. "Social network ordering based on communities to reduce cache misses". In: *Revue ARIMA* Volume 24 - 2016-2017 - Special issue CRI 2015 (May 2017).

[8] T. M. Nguélé, M. Tchuente, and J. Méhaut. "Using Complex-Network Properties for Efficient Graph Analysis". In: *Parallel Computing is Everywhere, Proceedings of the International Conference on Parallel Computing, ParCo 2017, 12-15 September 2017, Bologna, Italy*. 2017, pages 413–422.

[9] T. M. Nguélé. *DSL for Social Network Analysis On Multicore Architecture*. Sept. 2018.

[10] T. Messi Nguélé and J.-F. Méhaut. "Applying Data Structure Succinctness to Graph Numbering For Efficient Graph Analysis". In: (2021).

A ANNEX:PROVING THAT D IS A DISTANCE

Proof:. In other to show that D is a distance, we prove that it respects the four properties of a distance. Let N be the set of nodes:

1. $\forall a, b \in N, D(a, b) \geq 0$.
   This is because $|\Gamma(a)| + |\Gamma(b)| \geq 2|\Gamma(a) \cap \Gamma(b)|$
2. $\forall a, b \in N, D(a, b) = 0 \iff a = b$.
   In fact, when $a = b$, $|\Gamma(a)| + |\Gamma(b)| = 2|\Gamma(a) \cap \Gamma(b)|$
3. $\forall a, b \in N, D(a, b) = D(b, a)$
   This is because, $|\Gamma(a)| + |\Gamma(b)| = |\Gamma(b)| + |\Gamma(a)|$ and $|\Gamma(a) \cap \Gamma(b)| = |\Gamma(b) \cap \Gamma(a)|$
4. $\forall a, b, c \in N, D(a, c) \leq D(a, b) + D(b, c)$

For this last case, we should show that,

$$|\Gamma(a)| + |\Gamma(c)| - 2|\Gamma(a) \cap \Gamma(c)| \leq |\Gamma(a)| + |\Gamma(b)| - 2|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b)| + |\Gamma(c)| - 2|\Gamma(b) \cap \Gamma(c)|$$

Now we permute the elements of left and rigth and we also change the signe:

$$|\Gamma(a)| + |\Gamma(b)| - 2|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b)| + |\Gamma(c)| - 2|\Gamma(b) \cap \Gamma(c)| \geq |\Gamma(a)| + |\Gamma(c)| - 2|\Gamma(a) \cap \Gamma(c)|$$

the $|\Gamma(a)|$ and the $|\Gamma(c)|$ will cancel out and the $|\Gamma(b)|$ will add up;

We also pass $-2|\Gamma(a) \cap \Gamma(c)|$ on the other side of the inequality in order to have 0 on one side. Thus, we have:

$2|\Gamma(b)| - 2|\Gamma(a) \cap \Gamma(b)| - 2|\Gamma(b) \cap \Gamma(c)| + 2|\Gamma(a) \cap \Gamma(c)| \geq 0$

$\iff 2[|\Gamma(b)| - |\Gamma(a) \cap \Gamma(b)| - |\Gamma(b) \cap \Gamma(c)| + |\Gamma(a) \cap \Gamma(c)|] \geq 0$

$\iff |\Gamma(b)| - |\Gamma(a) \cap \Gamma(b)| - |\Gamma(b) \cap \Gamma(c)| + |\Gamma(a) \cap \Gamma(c)| \geq 0$

We group those who have the negative sign on one side and we have:

$|\Gamma(b)| + |\Gamma(a) \cap \Gamma(c)| - |\Gamma(a) \cap \Gamma(b)| - |\Gamma(b) \cap \Gamma(c)| \geq 0$

$\iff |\Gamma(b)| + |\Gamma(a) \cap \Gamma(c)| - [|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b) \cap \Gamma(c)|] \geq 0$

So to show that $D(a,b) \leq D(a,b) + D(b,c)$, it suffices to show that $|\Gamma(b)| + |\Gamma(a) \cap \Gamma(c)| - [|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b) \cap \Gamma(c)|] \geq 0$.

Let us show that: $|\Gamma(b)| + |\Gamma(a) \cap \Gamma(c)| - [|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b) \cap \Gamma(c)|] \geq 0$

$|\Gamma(b)| + |\Gamma(a) \cap \Gamma(c)| - [|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b) \cap \Gamma(c)|] \geq 0$

$\iff |\Gamma(b)| + |\Gamma(a) \cap \Gamma(c)| \geq [|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b) \cap \Gamma(c)|]$

With this inequality, the factors that can have different values are:

i^0) $|\Gamma(a) \cap \Gamma(c)| \leq |\Gamma(a)|$ If $|\Gamma(a)| < |\Gamma(c)|$ Or $\leq |\Gamma(c)|$ If $|\Gamma(c)| < |\Gamma(a)|$
ii^0) $|\Gamma(a) \cap \Gamma(b)| \leq |\Gamma(a)|$ If $|\Gamma(a)| < |\Gamma(b)|$ Or $\leq |\Gamma(b)|$ If $|\Gamma(b)| < |\Gamma(a)|$
iii^0) $|\Gamma(b) \cap \Gamma(c)| \leq |\Gamma(b)|$ If $|\Gamma(b)| < |\Gamma(c)|$ Or $\leq |\Gamma(c)|$ If $|\Gamma(c)| < |\Gamma(b)|$

For this inequality to no longer hold, the right side would have to be greater than the left side. The maximum cases for this to happen are:

<u>1st Case</u> : $|\Gamma(a) \cap \Gamma(b)| = |\Gamma(a)|$ et $|\Gamma(b) \cap \Gamma(c)| = |\Gamma(b)|$
<u>2nd Case</u>: $|\Gamma(a) \cap \Gamma(b)| = |\Gamma(a)|$ et $|\Gamma(b) \cap \Gamma(c)| = |\Gamma(c)|$
<u>3rd Case</u>: $|\Gamma(a) \cap \Gamma(b)| = |\Gamma(b)|$ et $|\Gamma(b) \cap \Gamma(c)| = |\Gamma(b)|$
<u>4th Case</u>: $|\Gamma(a) \cap \Gamma(b)| = |\Gamma(b)|$ et $|\Gamma(b) \cap \Gamma(c)| = |\Gamma(c)|$

From these different cases, we can determine the probable maximum value of $|\Gamma(a) \cap \Gamma(c)|$ contained in the left part of the inequality and see if the inequality will always be respected.

1.0.1 <u>1st case</u> : $(|\Gamma(a) \cap \Gamma(b)| = |\Gamma(a)| \;\; et \;\; |\Gamma(b) \cap \Gamma(c)| = |\Gamma(b)|$)

In this first case we have the following information:

$(|\Gamma(a)| < |\Gamma(b)|$ et $|\Gamma(b)| < |\Gamma(c)|) \implies |\Gamma(a)| \leq |\Gamma(b)| \leq |\Gamma(c)|$

$\implies |\Gamma(a) \cap \Gamma(c)| = |\Gamma(a)|$

The inequality $|\Gamma(b)| + |\Gamma(a) \cap \Gamma(c)| \geq [|\Gamma(a) \cap \Gamma(b)| + |\Gamma(b) \cap \Gamma(c)|]$

thus becomes $|\Gamma(b)| + |\Gamma(a)| \geq |\Gamma(a)| + |\Gamma(b)|$

And the inequality is therefore verified.

1.0.2 $\underline{2^{nd}\ case}$: $(|\Gamma(a) \cap \Gamma(b)| = |\Gamma(a)|\ \ et\ \ |\Gamma(b) \cap \Gamma(c)| = |\Gamma(c)|)$

Here $|\Gamma(a)| < |\Gamma(b)|$ and $|\Gamma(c)| < |\Gamma(b)| \implies |\Gamma(a)| \leq |\Gamma(b)| \geq |\Gamma(c)|$

In this case, we cannot directly conclude. We can however say that: $\exists\ q1, q2 \subset N\ /\ \ |\Gamma(b)| = |\Gamma(a)| + |q1|$ and $|\Gamma(b)| = |\Gamma(c)| + |q2|$

Thereby

$|\Gamma(a)| = |\Gamma(b)| - |q1| et |\Gamma(c)| = |\Gamma(b)| - |q2| \implies |\Gamma(a) \cap \Gamma(c)| = |\Gamma(a)| + |\Gamma(c)| - |\Gamma(a) \cup \Gamma(b)|$

or

$|\Gamma(a) \cup \Gamma(b)| = |\Gamma(b)|$ car $|\Gamma(b)| > |\Gamma(a)|$

The inequality can be written:

$|\Gamma(b)| + |\Gamma(a)| + |\Gamma(c)| - |\Gamma(b)| \geq |\Gamma(a)| + |\Gamma(c)|$

We note that the $|\Gamma(a)|$ and $|\Gamma(c)|$ will cancel on either side of the inequality and we will have:

$|\Gamma(b)| - |\Gamma(b)| \geq 0;$

The $|\Gamma(b)|$ will cancel out and we will therefore have a valid expression.

1.0.3 $\underline{3^{rd}\ case}$: $(|\Gamma(a) \cap \Gamma(b)| = |\Gamma(b)|\ \ et\ \ |\Gamma(b) \cap \Gamma(c)| = |\Gamma(b)|)$

For the 3^{ith} case, we have $|\Gamma(b)| < |\Gamma(a)|$ and $|\Gamma(b)| < |\Gamma(c)| \implies |\Gamma(a)| \geq |\Gamma(b)| \leq |\Gamma(c)|$

In this case, we cannot directly conclude. However We can say that: $\exists\ q3, q4 \subset N\ /\ \ |\Gamma(a)| = |\Gamma(b)| + |q3|$ and $|\Gamma(c)| = |\Gamma(b)| + |q4|$.

Thus, $\Gamma(a) \cap \Gamma(c) = (\Gamma(b) \cup q3) \cap (\Gamma(b) \cup q4) = \Gamma(b) \cup (q3 \cap q4)$

Hence $|\Gamma(a) \cap \Gamma(c)| = |\Gamma(b) \cup (q3 \cap q4)| = |\Gamma(b)| + |q3 \cap q4| - |\Gamma(b) \cap (q3 \cap q4)|$

The inequality therefore becomes:

$|\Gamma(b)| + |\Gamma(b)| + |q3 \cap q4| - |\Gamma(b) \cap (q3 \cap q4)| \geq |\Gamma(b)| + |\Gamma(b)|$

The $|\Gamma(b)|$ will all cancel out and we will have:

$|q3 \cap q4| - |\Gamma(b) \cap (q3 \cap q4)| \geq 0 \quad \iff \quad |q3 \cap q4| \geq |\Gamma(b) \cap (q3 \cap q4)|$

But by definition, $|q3 \cap q4| \geq |\Gamma(b) \cap (q3 \cap q4)$

The result is therefore valid, which allows us to verify our 3^{ieme} case.

1.0.4 $\underline{4^{th}\ case}$: $(|\Gamma(a) \cap \Gamma(b)| = |\Gamma(b)|\ \ et\ \ |\Gamma(b) \cap \Gamma(c)| = |\Gamma(c)|\ )$

For this last case, $|\Gamma(c)| < |\Gamma(b)|$ and $|\Gamma(b)| < |\Gamma(a)| \implies |\Gamma(a)| \geq |\Gamma(b)| \geq |\Gamma(c)|$

$$\implies |\Gamma(a) \cap \Gamma(c)| = |\Gamma(c)|.$$

The inequality therefore becomes: $|\Gamma(b)| + |\Gamma(c)| \geq |\Gamma(b)| + |\Gamma(c)|$

This gives us a valid result.

All cases could be tested and verified. So we have the proof that D is a distance.

B BIOGRAPHY

Regis Audran MOGO WAFO Computer Science Student/Phd in University of yaounde 1

Thomas Messi Nguélé Computer Science Doctor/PhD in University of yaounde 1

Xaviera Youth Kimbi Computer Science Doctor/PhD in University of yaounde 1