Scalable Computing Systems Laboratory

Project Report

Decentralized Representative Neighbourhood Construction for Machine Learning

By Maina Orchampt-Mareschal

Project advisor: Prof. Anne-Marie Kermarrec

Project supervisor: Dr. Erick Lavoie

January 7, 2022

1 Abstract

This report presents the exploration of the construction of a diverse neighbourhood for each node individually and has the goal to see if a similar convergence speed as D-cliques can be reached. Three algorithms have been implemented and are presented, along with the skew they give: the random topology, the bipartite topology, and a topology created with the SRP matching algorithm. The lowest skew obtained with those topologies equals 0.12 and is given by the random neighbourhood improved by the Swapping technique which is also presented. To compare it to the previous results obtained on topologies using cliques [1], it is the algorithm *Greedy swap* with fully connected cliques that gives the lowest skew, 0.06. Hence until now, the resulting skew of the construction without cliques is then proven to be higher than the one of construction with cliques. By simulating with the machine learning model with the same parameters as the one used to get the fastest speed of convergence with the *Greedy swap* algorithm, the quality of results obtained depends on which dataset is used. For a dataset with simple classes, the speed of convergence and the accuracy obtained for topologies without cliques are similar as for topologies with cliques. On the other hand, using a more complicated dataset, results seem to be less good for topologies without cliques. Indeed, the speed of convergence stays quite high for the topology with cliques while the one for the topologies without cliques decreases considerably.

Contents

List of Figures

List of Tables

2 Introduction

Recently, topologies using D-cliques [1] has been proven to be well adapted for Decentralized Federated Learning [2]. Indeed, since there is a positive correlation between the effect of local bias and the metric called skew, by reducing the skew of each clique, the overall skew of the topology is reduced and hence the effect of local bias is reduced too.

For now, the best results are obtained using an algorithm called *Greedy swap* [3]. The average skew obtained decreases with the increase of the size of the cliques and it starts to converge from a clique size of 25 and equals 0 from a clique size of 50. This algorithm works on a random topology of cliques and consists of swapping nodes between pairs of cliques.

As this kind of topology is based on unrealistic non-IID assumptions, it is interesting to dig the track of constructing a diverse neighbourhood for each node individually. This way, the overall skew of the topology would be reduced by minimising the skew of each neighbourhood instead of reducing the one of each clique. The goal of this report is then to study if a similar convergence speed as D-cliques can be obtained by only building diverse neighbourhoods for each node individually.

The first part of the report presents the implementation of those new topologies and the second part displays the results and their evaluation on machine learning models.

3 Representative Neighbourhood Topology Construction

In this section, the two algorithms as well as the problems encountered during their design are presented. The swapping technique which can be added directly to a neighbourhood to improve the topology by exchanging two nodes is also described.

3.1 Bipartite Neighbourhood Algorithm

The *Bipartite neighbourhood* algorithm, shown in Algorithm 1, is the first construction that has been thought of. The objective is to create a decentralized algorithm to create a neighbourhood topology by giving the possibility to each node of the set N to choose its preferred nodes amongst a random sample S. The preferred nodes of node i, $candidates_i$, are the nodes that would improve the skew of the neighbourhood of i, $skew_i$, if added, alone, to the neighbourhood of i, G_i. Before that, to verify if a node j from sample S can be added to the neighbourhood of i, four conditions need to be verified:

- the size of the neighbourhood of i, $|G_i|$, have to be lower than the limit of neighbours a node can have, M,
- the size of the neighbourhood of j, $|G_j|$, have to be lower than the limit of neighbours a node can have, M,
- j has to be different from i,
- and j should not be already present in the neighbourhood of i, G_i.

Once every node i of the set N have its preferred candidates, they are ordered in increasing skew of the neighbourhood of i to which j have been added, $skew_{ij}$.

Now comes the explanation of the name of the algorithm with the matching phase which was the trickiest part of the implementation. Indeed, with each node having an order list of

preferred candidates, the matching must be made. To do so, the nodes are separated into two categories: half of the nodes receive an active status while the other part receives a passive status. The active status permits a node to choose the first passive node in its list of candidates.

Also, the active nodes can only select passive nodes otherwise, an active node z could be elected and could elect a node which would lead to at least two new nodes in the neighbourhood of z for example. Since the candidates of z are chosen based on the computation of the skew at time $t0$, the second node added would maybe not even be part of the new candidates of node z to improve the skew of its neighbourhood since the latter has changed.

Finally, each time an active node selects a passive node, the neighbourhood and its skew for both nodes is updated which means that a new edge is created between the two nodes. A list of available passive nodes, $availablePassives$, is updated each time a passive has been selected by an active node so that a passive node can only be chosen once.

Algorithm 1 Bipartite Neighbourhood Algorithm

```
Require: maximum neighbours M, maximum creation steps K, a set of nodes N=1,2,...,n
  and the size of the random sample R
for k = 1, 2, ..., K do
    allCandidates ← []
    for i in N do
        candidates_i ← []
        if |G_i| < M then
            S ← sample(N, R)
            for j in S do
                if |G_j| < M AND j! = i AND j not in G_i then
                    G_ij ← {j} ∪ G_i
                    skew_ij ← skew(G_ij)
                    if skew_ij < skew(G_i) then
                        candidates_i append j
    Append candidate_i sorted by increasing order of skew to allCandidates

    activeNodes ← sample(N, N//2)
    availablePassives ← every node which is not active
    for i, candidates_i in allCandidates do
        if i is active then
            Filter candidates_i by only keeping passives
            if len(candidates_i) > 0 then
                for j in candidates_i do
                    if j in availablePassives then
                        Creation of an edge between i and j
                        Update of the neighbors and skew of the neighbourhoods of i and j
                        Remove j from availablePassives
                        break
return: the created neighbourhood
```

3.2 Algorithm Based on the Stable Roommates Problem

The resulting skew of the topology constructed with the *Bipartite neighbourhood* algorithm is lower than the one of a random topology but is still far above the skew obtained with the *Greedy swap* algorithm which uses fully connected D-cliques. Hence, a second algorithm, shown in Algorithm 2, has been designed to get a better skew than the one obtained with the first algorithm.

As stated in the previous subsection, the main problem of the implementation is the matching phase, hence this is the part that has to be rethought.

The Stable Roommates Problem (SRP) is the problem of finding a stable matching in which there is no pair of elements where both members prefer their partner in a different matching over their partner in the stable matching. It requires for each element an ordered list of all elements sorted by preferences: For example, having a set of node $P = 1, 2, 3, 4, 5$, each node have to give an ordered list of all the other nodes contained in P. Even if *node 1* does not want at all *node 3* since the skew of its neighbourhood would increases by adding it, *node 1* needs to have *node 3* in its list. This means that in each list of candidates for node i, it is likely that the last nodes are not even wanted by i.

Since the SRP requires exactly what we have, that is to say, a list of preferences for each node, the first part of the previous algorithm can be reused except for a small difference to satisfy the condition of the SRP: instead of constructing a list of preferences for each of the nodes in N from a sample, a list of preferences is only constructed for the nodes of a sample from the same sample.

This list of preferences is then given to special a python library [4] designed to solve the SRP and the stable matches, *matches*, is obtained.

For each pair of nodes in *matches*, the creation of an edge is not systematic. The three conditions need to be satisfied:

- the size of the neighbourhoods of both nodes have to be lower than the limit of neighbours a node can have, M,
- no edges is linking the two nodes already,
- and each of the two nodes contribute to the reduction of the skew of the neighbourhood of the other node.

In the case where all of them are respected, an edge is then created between the two nodes of the pair and their respective neighbourhoods and skews are updated.

3.3 Addition of Swaps

Since the resulting skew of both topologies created with the described algorithms is still really high, the swapping technique is introduced.

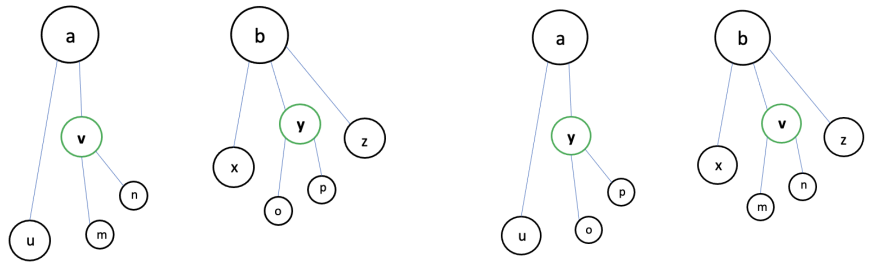
3.3.1 The Swapping Technique

The goal of this technique is to improve the skew of an overall topology by improving the skew of the neighbourhood of two nodes during each iteration. A swap involves four nodes. It is the action of exchanging two nodes, within the neighbourhoods of two others nodes.

Algorithm 2 SRP-based Algorithm

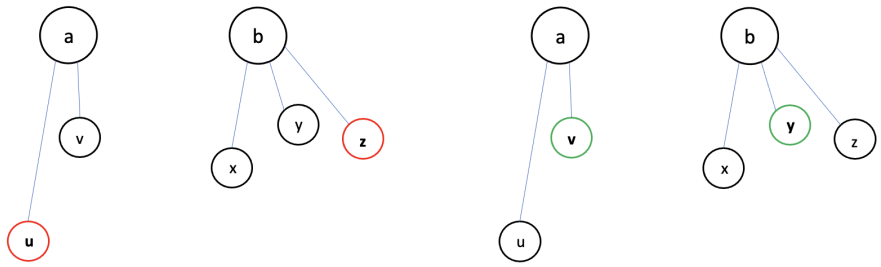
```
Require: maximum neighbours M, maximum creation steps K, a set of nodes N=1,2,...,n
and the size of the random sample R
for k = 1, 2, ..., K do
    S ← sample(N, R)
    allCandidates ← {}
    for i in S do
        for j in S do
            if i! = j then
                G_ij ← {j} ∪ G_i
                skew_ij ← skew(G_ij)
                candidates_i append j
        Append candidate_i sorted by increasing order of skew to allCandidates

    matches ← SRP(allCandidates)
    for i, j in matches do
        if i! = None AND j! = None then
            if |G_i| < M AND |G_j| < M AND j is not in G_i then
                G_ij ← {j} ∪ G_i
                skew_i ← skew(G_i)
                skew_ij ← skew(G_ij)
                skew_j ← skew(G_j)
                skew_ji ← skew(G_ji)
                if skew_ij < skew_i AND skew_ji < skew_j then
                    Creation of an edge between i and j
                    Update of the neighbors and skew of the neighbourhoods of i and j
return: the created neighbourhood
```



(a) Before swapping (b) After swapping

Figure 1: Swapping process



(a) First swap found (b) Best swap found

Figure 2: Comparison of the two swapping techniques

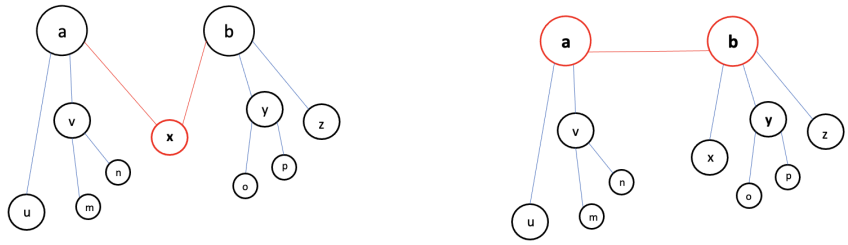
In Figure 1, nodes a and b are chosen randomly and their neighbourhood is checked to be improved. The swap between v and y is presented: Figure 1a shows the situation before the swap and Figure 1b shows the situation once the swap is done.

A swap is only done if it improves the skew of the neighbourhoods of the four nodes implied. In Figure 1, the swap between v and y is done if it improves the skew of the neighbourhoods of a, b, v and y. To verify that, the easiest way is to compare the sum of the four skews, and this is what is used in the implementation.

Two swapping techniques have been tried. The first one proceeds as follow: when traversing both neighbourhoods, the first possible swap is taken into account. The disadvantage with this technique is that if there is a better swap coming later while traversing the two neighbourhoods for this pair of nodes, it won't get the chance to be seen. That is why a second swapping technique is introduced. This time, it is the best swap that is taken into account. When traversing the two neighbourhoods simultaneously, it keeps in memory the best swap and applies the swap once all the possibilities have been verified to ensure it is the best one. Figure 2 shows an example of the two techniques: in Figure 2a, the nodes circled in red are the ones which would be done using the first technique, while in Figure 2b, the nodes circled in green are the ones which would be done using the first technique.

The second swapping technique has been chosen for the experiences done since it gives better results. Its implementation is shown in Algorithm 3.

Before checking if a swap will improve the skew of the topology, a swap has to be valid. Indeed, there are two cases when a swap would lead to an imbalance topology, with imbalanced



(a) The neighbourhoods of a and b share at least one nodes

(b) a and b are neighbors

Figure 3: Situations when no swap can be done

weights, and hence can't be done. Those two cases are presented in Figure 3.

The first one, shown in Figure 3a, is when a swap involves a node that is shared between the two neighbourhoods. By doing this swap, a or b would end up with twice the same node but since a node cannot be duplicated (as sets are used), an invalid edge would be created.

The second one, shown is Figure 3b, represents the situation when a and b are direct neighbors. Then, by iterating on the neighbourhood of a, node b could be swapped with a node of its own neighbourhood. This would lead to the creation of a duplicate of node b. Again, since a node cannot be duplicated, an invalid edge would be created.

Both situations can be avoided by complying with one condition: the intersection of the neighbourhoods of a and b has to be empty but this condition is too strict. To not lose all the possible valid swaps between two nodes that have a non-empty intersection of their neighbourhoods, the simultaneous iteration on both neighbourhoods is done, but only on the nodes which cannot create problems. The valid part of the neighbourhood of a is then the neighbourhood of a minus the neighbourhood of b and the valid part of the neighbourhood of b is then the neighbourhood of b minus the neighbourhood of a. This way, all the invalid swaps are avoided but no valid ones are lost.

3.3.2 Previous Algorithms Improved by Swaps

Now that a technique has been implemented to improve the skew of a diverse neighbourhood topology, it is interesting to study its impact on the topologies created by the two algorithms seen previously as well as on a random topology. It has been found that the skews of the overall topologies are much lower than the ones obtained previously and the results are shown in the next section. The two algorithms improved by the swapping technique are then called *Bipartite neighbourhood swap* and *SRP swap* and are both composed of two parts. The first one is the creation part which is the creation of the topology using the previous algorithms, Algorithm 1 and Algorithm 2 respectively. The second one is the swapping part which is the improvement of the created topology by the swapping technique. Both have a different number of steps, the *creation steps* and the *swapping steps* respectively. The same update has been done on the random algorithm which simply creates a random diverse neighbourhood. The new algorithm is then called *Random swap* and is also separated into two parts, the creation part and the swapping part.

Algorithm 3 Swapping Algorithm

```
Require: maximum swapping steps W, a set of nodes N=1,2,...,n, and the corresponding neighbourhood G
for w = 1, 2, ..., W do
    pair ← sample(N, 2))
    skew_a ← skew(G_a)
    skew_b ← skew(G_b)
    if |G_a ∪ G_b| > 0 then
        g_a ← G_a - G_b
        g_b ← G_b - G_a
    else
        g_a ← G_a
        g_b ← G_b
    for i in g_a do
        for j in g_b do
            if i != a AND j != b then
                initialSumSkews ← skew_a + skew_b + skew(G_i) + skew(G_j)
                G'_a ← replace i by j in G_a
                G'_b ← replace j by i in G_b
                G'_i ← replace a by b in G_i
                G'_j ← replace b by a in G_j
                skew'_a ← skew(G'_a)
                skew'_b ← skew(G'_b)
                skew'_i ← skew(G'_i)
                skew'_j ← skew(G'_j)
                newSumSkews ← skew'_a + skew'_b + skew'_i + skew'_j
                if newSumSkews < initialSumSkews then
                    if bestSwap is empty then
                        Initialisation of bestSwap with i, j, the updated neighbourhood of a,
                        b, i, j and their respective skews as well as the sum of the skews
                    else
                        if newSumSkews < bestSwap[sumSkews] then
                            Update bestSwap with the new swap
    if bestSwap is not empty then
        Do the swap kept in memory and update the neighbourhoods of the nodes involved
return: the improved neighbourhood
```

4 Evaluation

The results obtained are shown in this section. The first focus is on the topologies themselves and especially on their skew distribution and their skew convergence through a parameter study. The other interesting aspect is to understand if the difference in skew between the different algorithms is significant for the training machine learning model. The performance of the model of the different topologies is analysed by looking at the convergence speed and the model accuracy.

4.1 Parameter Study

For all the experiments, most of the parameters are fixed to stay consistent with what have been done previously [1]. The number of nodes used is fixed to 100 nodes, the maximum neighbours a node can have is 10, the number of local shards is fixed to 2 and the shard size is fixed to 248. The main parameter is then the size of the random sample used in both the creation of the bipartite topology and the SRP topology. Indeed, depending on it, the number of creation steps and the number of swapping steps are kind of fixed since they are the ones when the convergence of each part (creation and improvement) has been reached. For the *Random swap* algorithm, as no random sample is used for the creation, the number of creation steps the number of swapping steps are fixed.

For each of the *Bipartite neighbourhood* algorithm, the *SRP* algorithm, the *Bipartite neighbourhood swap* algorithm and the *SRP swap* algorithm, three different size of random sample have been used:

- 4 nodes as lower limit case since it represents approximately the logarithm of the number of nodes used ($\ln(100)$),
- 20 nodes since the algorithm *Greedy swap* works well with two cliques of size 10 which is 20 nodes in total,
- and 100 nodes as upper limit case since the number of nodes used is 100.

The number of creation steps required to reach the convergence has been found in the parameter study of the two algorithms *Bipartite neighbourhood* and the *SRP*. Those creation steps were then used in the parameter study of the *Bipartite neighbourhood swap* algorithm and the *SRP swap* algorithm to find the corresponding number required swapping steps.

One experiment has been conducted for each size of random sample for each algorithm and the results of the parameter study are shown in Table 1. For each one, the following values are given:

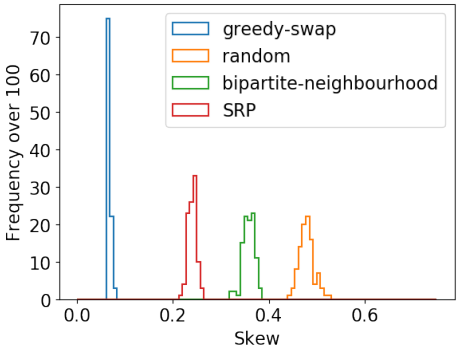
- the average number of neighbours which can be 11 at maximum since it is the maximum number of neighbours a node can have plus itself,
- the average skew before swapping,
- the final average skew which is the skew after swapping,
- and the number of swaps performed.

Bold lines show the experiment that is the most appropriate for each size of random sample for each algorithm. For example, the topology created with the *Bipartite neighbourhood* algorithm with a random sample of size 4 converges from 100 steps and a skew of 0.3319 is obtained. When this same topology is created but is also improved by swaps, using the *Bipartite neighbourhood swap* algorithm, 3000 swapping steps are needed to reach the convergence of the topology and the final skew obtained is 0.1337. The lines highlighted in green represent the best combination (the fastest) to obtain the best skew for each of the three algorithms improved by swaps.

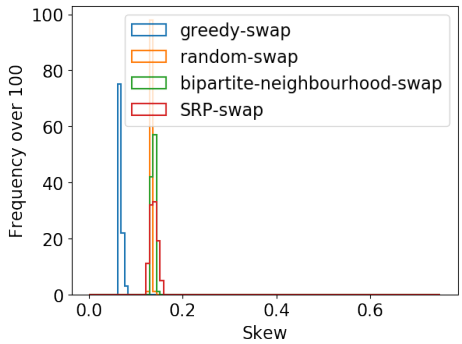
The bold experiments were run a hundred times to obtain results statistically significant. Two aspects of the algorithms are interesting to compare: the distribution of the skew and the speed of convergence of the skew.

	On 1 experiment						
	Sample size	Creation steps	Swapping steps	Average neighbours (11 max)	Average skew before swapping	Final average skew	# Swaps
Bipartite-neighbourhood	4	40		10.74		0.3394	
	4	**100**		**10.96**		**0.3319**	
	4	500		10.98		0.3316	
	20	**20**		**10.86**		**0.3047**	
	20	100		10.92		0.3061	
	20	500		10.92		0.3061	
	20	1000		10.92		0.3061	
	100	10		9.38		0.2883	
	100	**20**		**10.9**		**0.2611**	
	100	50		10.92		0.2613	
	100	100		10.92		0.2613	
	100	500		10.92		0.2613	
Bipartite-neighbourhood-swap	4	100	500	10.96	0.3319	0.1756	242
	4	100	1000	10.96	0.3319	0.1534	321
	4	100	2000	10.96	0.3319	0.1393	376
	4	**100**	**3000**	**10.96**	**0.3319**	**0.1337**	**412**
	4	100	5000	10.96	0.3319	0.1297	436
	4	100	10000	10.96	0.3319	0.1254	469
	20	20	500	10.86	0.3047	0.1735	209
	20	20	1000	10.86	0.3047	0.1528	273
	20	**20**	**3000**	**10.86**	**0.3047**	**0.1361**	**337**
	20	20	5000	10.86	0.3047	0.1316	369
	20	20	10000	10.86	0.3047	0.1294	386
	100	20	500	10.9	0.2611	0.1746	177
	100	20	1000	10.9	0.2611	0.1604	228
	100	**20**	**5000**	**10.9**	**0.2611**	**0.1378**	**325**
	100	20	10000	10.9	0.2611	0.1342	363
SRP	4	700		7.52		0.2951	
	4	1000		8.28		0.2614	
	4	**3000**		**8.98**		**0.2337**	
	4	5000		9.04		0.2303	
	4	10000		9.1		0.2285	
	20	100		6.22		0.2587	
	20	**200**		**6.58**		**0.2414**	
	20	500		6.6		0.2407	
	100	500		5.3		0.2376	
	100	20		5.3		0.2376	
	100	**10**		**5.24**		**0.2401**	
SRP-swap	4	3000	1000	8.98	0.2337	0.1632	145
	4	**3000**	**5000**	**8.98**	**0.2337**	**0.1321**	**243**
	4	3000	10000	8.98	0.2337	0.1285	266
	20	200	3000	6.58	0.2414	0.1678	125
	20	200	5000	6.58	0.2414	0.1597	152
	20	**200**	**10000**	**6.58**	**0.2414**	**0.1458**	**191**
	20	200	200000	6.58	0.2414	0.1377	214
	20	200	300000	6.58	0.2414	0.1368	220
	100	10	1000	5.24	0.2401	0.2121	34
	100	10	5000	5.24	0.2401	0.1833	81
	100	**10**	**10000**	**5.24**	**0.2401**	**0.1746**	**98**
	100	10	20000	5.24	0.2401	0.1694	110
	100	10	30000	5.24	0.2401	0.1686	115
Random-swap		5	1000	11	0.4538	0.1545	424
		5	**5000**	**11**	**0.4538**	**0.1295**	**536**
		5	10000	11	0.4538	0.1273	552

Table 1: Results of the parameter study



(a) Comparison of the skew distributions of the creation algorithms



(b) Comparison of the skew distributions of the algorithms once improved by the swapping technique

Figure 4: Skew distributions

Until now, both the creation and the improvement of the topology construction were taken into account hence the analysis of these two aspects continues this way. A sample of size 4 is used for the rest of the evaluation.

The distribution of the skew for the creation algorithms are shown in Figure 4a It is clear that without the improvement of those topologies by the swapping technique, the final skew is far above the one obtained with the *Greedy swap* algorithm. Nevertheless, both of the creation algorithms have an average final skew lower than the *Random* algorithm.

Once improved by the swapping techniques, the skews obtained is much closer to the one of the *Greedy swap* algorithm. In contrast to the previous results, before improvements, where the bipartite neighbourhood has a better skew than the SRP neighbourhood which has a better skew than a random neighbourhood, the order has now changed. The random topology improved by swap as well as the SRP topology improved by swap reach now the lowest skew after the *Greedy swap* algorithm. Indeed, they reach a skew of approximately 0.12 when the *Greedy swap* algorithm reaches a skew of 0.06. Results are shown in Figure 4b.

About the speed of convergence of the skew, the results are shown in Figure 5. Each curve includes both the creation part (with a certain number of creation steps) and the improvement part (with a certain number of swapping steps). Since we are analysing the experiments using a sample size of 4, the number of creation steps and the number of swapping steps can be found in the lines highlighted in green in Table 1. The *Bipartite neighbourhood swap* algorithm begins with 100 creation steps followed by 3000 swapping steps which is a total of 3100 steps. Then, the *SRP swap* algorithm begins with 3000 creations steps followed by 5000 swapping steps which is a total of 8000 steps. Finally, the *Random swap* algorithm begins with only 5 creation steps followed by 5000 swapping steps which is a total of 5005 steps. The change between the creation and the improvement part is clear for the three curves on the graph. Indeed, the pattern of the curves changes at 3000 steps for the *Bipartite swap* algorithm, at 100 steps for the *SRP swap* algorithm and at 5 steps for the *Random swap* algorithm.

The *Greedy swap* algorithm is again present on the graph to compare with the best topology involving cliques and it can be seen that it converges much faster than the other algorithm. Also, even if the *SRP swap* algorithm and the *Random swap* algorithm reach a value of skew not so far above, they both need a lot more steps to reach the convergence. Indeed, when

the *Greedy swap* algorithm reach the convergence with a skew of 0.06 (1000 steps), the *SRP swap* algorithm is still in its creation phase with a skew higher than 0.25 and the *Random swap* algorithm is already in its swapping phase but still has a skew higher more or less equal to 0.20.

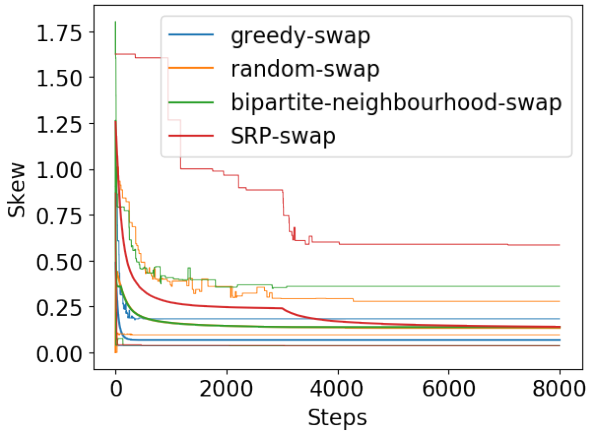


Figure 5: Comparison of the speeds of convergence of the skew of the algorithms

4.2 Machine Learning Application

Until now, three new algorithms have been studied: the *Bipartite neighbourhood swap* algorithm, the *SRP swap* algorithm, and the *Random swap* algorithm. The lower skew reached by them is still higher than the one obtained with the *Greedy swap* algorithm which uses cliques.

The focus is now on studying if the difference in skew between all those algorithms is significant for training machine learning models. To do the comparison, the three experiments which give the best results on the topology constructed by the *Greedy swap* algorithm (see parameters in Table 2) have been done on the three topologies presented in the previous section. The same parameter has been used unless the parameter *clique gradient* which has been set to *false* since the three topologies do not have cliques.

Dataset	**Model**	**Clique gradient**	**Learning rate**	**Momentum**	**Epochs**
MNIST	linear	true	0.1	0.0	100
CIFAR	gn lenet	true	0.002	0.0	100
CIFAR	gn lenet	true	0.002	0.9	100

Table 2: Parameters of the 3 simulations which give the best results for the *Greedy swap* algorithm

Hence the first experiment done on 100 epochs, on the topologies created by the *Bipartite swap* algorithm, the *SRP swap* algorithm, and the *Random swap* algorithm has been done on the *MNIST* dataset using a linear model, no clique gradient, a learning rate of 0.1 and a momentum of 0.0. The comparison of the performance of this model is shown in Figure 6. The topology created by the *Bipartite neighbourhood swap* algorithm and by the *Random swap* algorithm give results as accurate as the one given by the *Greedy swap* algorithm with an accuracy of approximately 90% and their speed of convergence is also very similar. On another side, the model works less well on the topology created by the *SRP swap* algorithm since its

speed of convergence is a bit lower but its accuracy is still close to the accuracy given by the other ones.

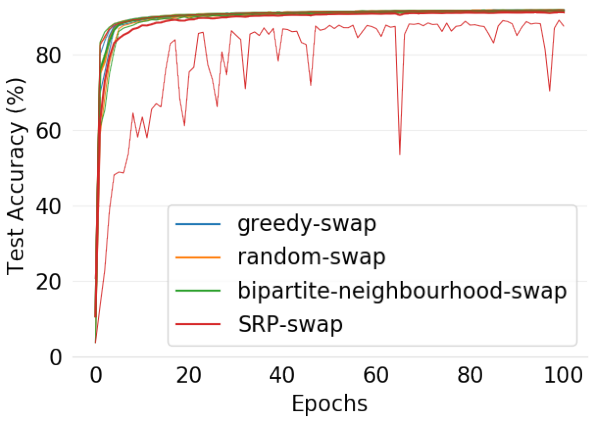


Figure 6: Performance of the model using *MNIST* of the different topologies on 100 epochs

Then, two other simulations were done on 100 epochs, on the *CIFAR* dataset with a learning rate of 0.002. The first one uses a momentum of 0.0 to see the result with *CIFAR* for no momentum too. The comparison of the performance of this model on the different topologies is shown in Figure 7. This time, the accuracy given by the different topologies varies significantly. Indeed, the results given by the *Greedy swap* algorithm are better than the best ones of the three other algorithms with an accuracy of approximately 65% and its speed of convergence is way higher too. The *Bipartite neighbourhood swap* algorithm gives results similar to the *Random swap* algorithm with more than 40% of accuracy and they have more or less the same speed of convergence. Finally, the *SRP swap* algorithm gives again the worst results with a test accuracy of around 30% and a lower speed of convergence.

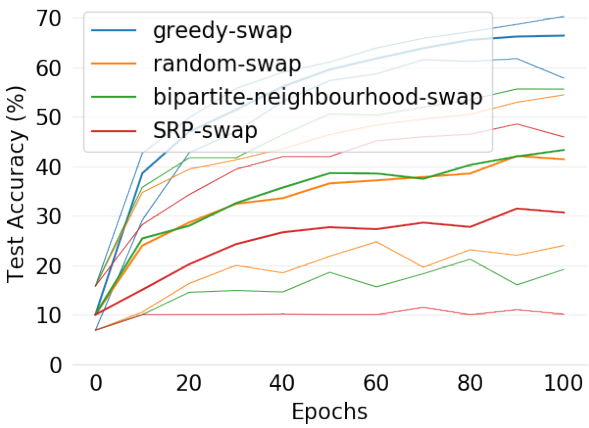


Figure 7: Performance of the model using *CIFAR* of the different topologies without momentum on 100 epochs

Finally, the second experiment on the *CIFAR* dataset uses a momentum of 0.9 since it is the one which permits to obtain the best result on the topology constructed by the *Greedy swap* algorithm. The comparison of the performance of this model on the different topologies is shown in Figure 8. The speed of convergence given by the *Greedy swap* algorithm is even greater than before, and the test accuracy obtained is above 70%. But this time, the three topologies without cliques give the same speed of convergence as well as the same accuracy which is lower than before (less than 30%).

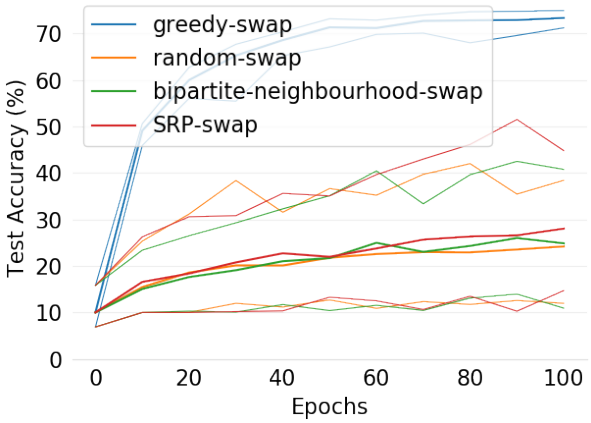


Figure 8: Performance of the model using *CIFAR* of the different topologies with a momentum of 0.9 on 100 epochs

5 Conclusion

Topologies using D-cliques have indeed been proven to be well adapted for Decentralized Federated Learning, but if a similar convergence speed can be obtained with a diverse neighbourhood for each node individually, then no cliques would be needed. Three constructions of a diverse neighbourhood for each node individually have been studied: a random diverse neighbourhood, a bipartite neighbourhood and a neighbourhood constructed using the SRP matching algorithm. The interesting results were obtained once those three topologies have been improved by the swap technique but still have a higher skew than the topologies using cliques. Indeed, the topology without cliques giving the lowest skew is the random swap algorithm and the skew obtained equals 0.12 while the lowest skew obtained by topologies with cliques equals 0.06. Despite this difference in skew, it is shown that the machine learning model on the dataset *MNIST* seems to give results as good as the topologies using cliques for the topologies created by the *Bipartite neighbourhood* swap algorithm and for the *Random swap* algorithm. Indeed the speed of convergence and the accuracy are similar. On the more complex dataset, *CIFAR*, the results are not as good on topologies without cliques. The speed of convergence is much lower and the accuracy is around 30% while it is above 70% with cliques. Hence, it seems that whether using a topology with or without cliques depend on the complexity of the dataset used. When working with a complex dataset, continuing building topologies with cliques seem to be a more interesting decision but when using a simpler dataset, no clique are needed, and no complex topologies are needed too since a simple random topology improved by swap gives enough good results.

References

[1] A. Bellet, A.-M. Kermarrec, and E. Lavoie. *D-Cliques: Compensating NonIIDness in Decentralized Federated Learning with Topology*. 2021. URL: https://arxiv.org/abs/2104.07365. (accessed: 20.10.2021).

[2] P. Kairouz. *Advances and Open Problems in Federated Learning*. 2019. URL: https://arxiv.org/abs/1912.04977. (accessed: 20.10.2021).

[3] P. Vasquez and A. Senderovich. "D-Cliques Construction". In: (2021).

[4] Henry Wilde. *A package for solving matching games*. 2020. URL: https://pypi.org/project/matching/. (accessed: 07.11.2021).