Scalable Anti-KNN: Decentralized Computation of k-Furthest-Neighbor Graphs with HyFN

Simon Bouget[1], David Bromberg[1,2], François Taïani[1,2], Anthony Ventresque[3]

[1] Université de Rennes 1 - IRISA, [2] ESIR, Rennes, France, [3] Lero@UCD, School of Computer Science, University College Dublin, Ireland
{simon.bouget, david.bromberg, francois.taiani}@irisa.fr,
anthony.ventresque@ucd.ie

Abstract. The decentralized construction of k-Furthest-Neighbor graphs has been little studied, although such structures can play a very useful role, for instance in a number of distributed resource allocation problems. In this paper we define KFN graphs; we propose HyFN, a generic peer-to-peer KFN construction algorithm, and thoroughly evaluate its behavior on a number of logical networks of varying sizes.

1 Motivation

k-Nearest-Neighbor (KNN) graphs have found usage in a number of domains, including machine learning, recommenders, and search. Some applications do not however require the k closest nodes, but the k most dissimilar nodes, what we term the k-Furthest-Neighbor (KFN) graph.

Virtual Machines (VMs) placement —i.e. the (re-)assignment of workloads in virtualised IT environments— is a good example of where KFN can be applied. The problem consists in finding an assignment of VMs on physical machines (PMs) that minimises some cost function(s) [27]. The problem has been described as one of the most complex and important for the IT industry [3], with large potential savings [20]. An important challenge is that a solution does not only consist in packing VMs onto PMs — it also requires to limit the amount of interferences between VMs hosted on the same PM [31]. Whatever technique is used (e.g. clustering [21]), interference aware VM placement algorithms need to identify complementary workloads — i.e. workloads that are dissimilar enough that the interferences between them are minimised. This is why the application of KFN graphs would make a lot of sense: identifying quickly complementary workloads (using KFN) to help placement algorithms would decrease the risks of interferences.

The construction of KNN graphs in decentralized systems has been widely studied in the past [17, 30, 4, 14]. However, existing approaches typically assume a form of "likely transitivity" of similarity between nodes: if A is close to B, and B to C, then A is likely to be close to C. Unfortunately this property no longer holds when constructing KFN graphs. As a result, these approaches, as demonstrated in the remainder of the paper, are not working anymore when applied to this new problem.

Algorithm 1: Greedy decentralized KNN algorithm executing at node p

each round do
  $q \leftarrow$ one random neighbor from $\Gamma(p)$
  send $\langle \text{PUSH}, \Gamma(p) \cup \{p\} \rangle$ **to** q ; **request** $\Gamma(q)$ **from** q ▷ *push - pull*
  $cand \leftarrow \Gamma(p) \cup \Gamma(q) \cup \{r \text{ random nodes}\} \setminus \{p\}$
  $\Gamma(p) \leftarrow \text{argtop}^k_{g \in cand}\big(\text{sim}(p, g)\big)$

on receiving $\langle \text{PUSH}, \Gamma' \rangle$ **do**
  $cand \leftarrow \Gamma(p) \cup \Gamma' \setminus \{p\}$
  $\Gamma(p) \leftarrow \text{argtop}^k_{g \in cand}\big(\text{sim}(p, g)\big)$

To address this problem, this paper proposes HyFN (standing for *Hybrid KFN*, pronounced *hyphen*), an hybrid decentralized approach for the decentralized construction of k-furthest-neighbor graphs. We show that HyFN is able to construct a KFN graph with 3200 nodes in less than 17 rounds, when a traditional greedy approach is unable to converge. We also show that our proposal is highly scalable, with a convergence time evolving in $O(log(n))$ for larger graphs.

The remainder of this paper is organized as follows: we first discuss some background about k-nearest-neighbor (KNN) graphs and their decentralized construction in peer-to-peer networks, before presenting our intuition for the construction of a k-furthest-neighbor graph (KFN) in Section 2. In Section 3, we describe in more detail HyFN and its variants. We evaluate our approach in Section 4, discuss related work in Section. 5 and conclude in Section 6.

2 Decentralized Construction of a KFN graph

2.1 Background: Decentralized k-nn Graph Construction

The problem of constructing a k-furthest-neighbor (KFN) graph can be seen as a variant of a k-nearest-neighbor (KNN) graph construction that uses an opposed similarity.

A large body of works have been proposed to construct KNN graphs in decentralized systems, with applications ranging from recommendation [4, 14, 19], to search [13], to news dissemination [6]. In such systems, nodes (e.g. representing a user) can connect to each other using point-to-point networking, but only maintain a small partial view of the rest of the system, typically a small-size neighborhood of other nodes. Each node also stores a profile (e.g. a user's browsing history), and uses a peer-to-peer epidemic protocol [1, 4, 30, 17] to converge towards an optimal neighborhood, i.e. a neighborhood containing the k most-similar other nodes in the system according to some similarity metric on profiles (e.g. cosine similarity, or Jaccard's coefficient).

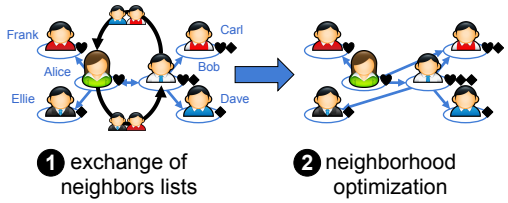


Fig. 1. A round of greedy decentralized KNN construction

The principle of a typical P2P protocol for KNN graph construction [9, 30] is shown in Algorithm 1, in its push-pull variant[1]. Starting from a random neighborhood, individual nodes repeatedly select a random neighbor q (line 2), exchange their current neighborhood with that of q (noted $\Gamma(q)$, line 4), and use the gained information to select more similar neighbors (line 5)[2]. Similarly, when receiving a new neighborhood pushed to them, nodes update their local view with the new nodes they have just heard of (lines 6-8). The intuition behind this greedy procedure is that if A is similar to B, and B to C, C is likely to be similar to A as well. To avoid local minima, this greedy procedure is often complemented with a few random peers (returned by a *peer sampling service* [18], tuned with parameter r at line 4).

This mechanism is illustrated in Figure 1. In this example, node *Alice* is interested in hearts (*Alice*'s profile), and is currently connected to *Frank*, and to *Ellie*. During this round, Alice selects *Bob* as her exchange partner. After exchanging her neighbors list with *Bob*, *Alice* finds out about *Carl*, who appears to be a better neighbor than *Ellie*. As such, *Alice* replaces *Ellie* with *Carl* in her neighborhood. Similarly *Bob* detects that *Ellie* is a better neighbor than *Alice*, and drops *Alice* in favor of *Ellie*.

2.2 Moving to Decentralized k-furthest-neighbor Graph Construction

Algorithm 1 can be easily adapted to compute a decentralized k-furthest-neighbor (KFN) graph by using a negative similarity at line 5:

$$\Gamma(p) \leftarrow \underset{g \in cand}{\overset{k}{\operatorname{argtop}}} \big(-\operatorname{sim}(p, g) \big) \tag{1}$$

Unfortunately, with this modification, one of the key premises of Algorithm 1 disappears: the far neighbors of a far neighbor are not so likely to be interesting

[1] The presented model is close to the *Vicinity* algorithm [30], but variations exist, most notably the T-Man algorithm [17], which buffers and selects nodes differently.

[2] argtop^k returns a k-tuple of nodes that maximizes the similarity function $\operatorname{sim}(p, -)$. Said differently, argtop^k generalizes the concept of argument of the maximum (argmax for short) to the k top values of a function over a finite discrete set.

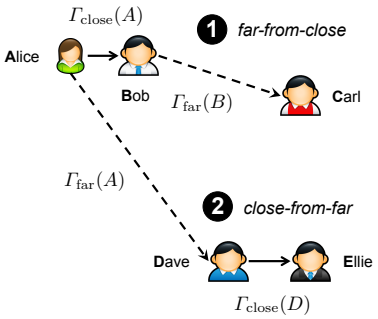


Fig. 2. The two heuristics we propose to construct a KFN graph

candidates to construct a KFN graph. Said differently, if A is far from B, and B far from C, this does not imply that A is far from C (or further from C than any other node taken randomly in the dataset).

Starting from this observation, we propose instead to use a *dual strategy* that constructs an *intermediate KNN graph* in order to construct *a final KFN graph*. In our approach, each node p maintains two views containing k nodes each: $\Gamma_{\text{close}}(p)$ and $\Gamma_{\text{far}}(p)$.

$\Gamma_{\text{close}}(p)$ uses the algorithm shown in Algorithm 1 to converge towards the k most similar other nodes in the system. $\Gamma_{\text{far}}(p)$ employs two greedy optimization heuristics that exploits $\Gamma_{\text{close}}(p)$ to progressively discover the k furthest neighbors from p. The intuition behind these two heuristics (shown in Figure 2 in the case of the node *Alice*) is as follows:

- The first heuristic (termed *far-from-close* and labeled **1** in the figure) requests the "far neighborhood" $\Gamma_{\text{far}}(B)$ of a node *Bob* found in *Alice*'s "close neighborhood" $\Gamma_{\text{close}}(A)$. The idea is that if *Bob* is close to *Alice*, then nodes that are far from *Bob* (such as *Carl* in Figure 2) will also be far from *Alice*.
- The second heuristic (termed *close-to-far* and labeled **2** in the figure) requests the "close neighborhood" $\Gamma_{\text{close}}(D)$ of a node *Dave* found in *Alice*'s "far neighborhood" $\Gamma_{\text{far}}(A)$. The idea is that if *Dave* is far from *Alice*, then nodes that are close to *Dave* (such as *Ellie* in Figure 2) will also be far from *Alice*.

In the following we present HyFN, a general algorithm that combines the two heuristics described above in various measures.

3 Algorithms

3.1 General Framework

Algorithm 2 provides an overview of the approach we propose, termed HyFN, as executed by Node p. For a fair comparison with a traditional greedy approach,

Algorithm 2: HyFN: A generic algorithm to implement a KFN computation, executing at node p

```
1  Init: For each p, Γ_close(p) and Γ_far(p) are heaps of size k, initialized as empty.

2  each round do
3      with probability α do
4          updateCloseView()
5      otherwise
6          updateFarView()

7  procedure updateCloseView() is
8      q ← one random neighbor from Γ_close(p)
9      send ⟨close, Γ_close(p) ∪ {p}⟩ to q ; request Γ_close(q) from q   ▷ push-pull
10     cand_close ← Γ_close(p) ∪ Γ_close(q) ∪ {r random nodes} \ {p}
11     Γ_close(p) ← argtop^k_{g∈cand_close} (sim(p, g))

12 on receiving ⟨close, Γ'_close⟩ do
13     cand_close ← Γ_close(p) ∪ Γ'_close \ {p}
14     Γ_close(p) ← argtop^k_{g∈cand_close} (sim(p, g))

15 procedure updateFarView() is
16     cand_far ← Γ_far(p) ∪ farCandidatesXX(p) ∪ {r random nodes}
17     Γ_far(p) ← argtop^k_{g∈cand_far} ( − sim(p, g))
```

we limit ourselves to *one* push-pull exchange per round and per node (as in Algorithm 1). This limitation is key to properly assess the interest of our approach: an algorithm that exchanges more information is naturally advantaged against its more frugal competitors. It would for instance be unfair to compare an algorithm using multiple push-pull exchanges to maintain multiple views against Algorithm 1, as such an algorithm would be more costly in terms of network traffic.

To ensure only one push-pull exchange is performed per round we use the construct **with probability** α **do** .. **otherwise** at line 3. This construct executes with a given probability (here α) the first statement, and with a probability $(1 - \alpha)$ the second. In this particular case, Algorithm 2 randomly alternates between invoking updateCloseView() at line 4, and invoking updateFarView() at line 6. Both procedures (discussed below), only generate one network exchange per node and per round, thus enforcing our communication limit. updateCloseView() maintains $\Gamma_{\text{close}}(p)$, p's close neighborhood, while updateFarView() uses $\Gamma_{\text{close}}(p)$ to construct $\Gamma_{\text{far}}(p)$. The parameter α (contained in $[0, 1]$) measures out how much effort each node will spend on $\Gamma_{\text{close}}(p)$ rather than $\Gamma_{\text{far}}(p)$.

updateCloseView(), shown at lines 7-11, uses Algorithm 1 (discussed in Section 2.1) to construct $\Gamma_{\text{close}}(p)$. updateFarView() depends on a pluggable

Algorithm 3: A far-from-close strategy to select far candidates (at p)

```
1 procedure farCandidatesFarFromClose(node p) is
2     q_close ← one random neighbor from Γ_close(p)
3     send ⟨FAR, Γ_far(p)⟩ to q_close ; request Γ_far(q_close) from q_close      ▷ pull
4     return Γ_far(q_close)
```

Algorithm 4: A close-to-far strategy to select far candidates (at p)

```
1 procedure farCandidatesCloseToFar(node p) is
2     q_far ← one random neighbor from Γ_far(p)
3     send ⟨FAR, Γ_close(p) ∪ {p}⟩ to q_far ; request Γ_close(q_far) from q_far   ▷ pull
4     return Γ_close(q_far)
```

Algorithm 5: Reception of a FAR push message (at p)

```
1 on receiving ⟨FAR, Γ'_far⟩ do
2     cand_far ← Γ_far(p) ∪ Γ'_far
3     Γ_far(p) ← argtop^k_{g∈cand_far} ( − sim(p, g))
```

Algorithm 6: A mixed strategy to select far candidates (at node p)

```
1 procedure farCandidatesMixed(node p) is
2     with probability β do
3         return farCandidatesCloseToFar(p)
4     otherwise
5         return farCandidatesFarFromClose(p)
```

procedure farCandidatesXX(p), which exchanges potential new candidate nodes using a push-pull approach to update p's far neighborhood, $\Gamma_{\text{far}}(p)$ at line 16. The current far neighborhood of p, the nodes received by farCandidatesXX(p), and r random nodes are stored in the intermediate $cand_{\text{far}}$ variable (line 16). The k furthest nodes from $cand_{\text{far}}$ then become p's new far neighborhood (line 17; note the minus sign before $\text{sim}(p, g)$, in contrast to line 11). (We discuss the push part of the exchange just below.)

3.2 Instantiating the selection of far candidates

The pluggable method farCandidatesXX(p) can be instantiated in three different manners, with the procedures farCandidatesFarFromClose(p), farCandidatesCloseToFar(p) and farCandidatesMixed(p), shown in Algorithms 3, 4, and 6.

- farCandidatesFarFromClose(p) (Algorithm 3) implements the *far-from-close* strategy discussed in Section 2.2: the local node p first selects one of its close neighbors q_{close} (line 2), and returns the far neighbors of q_{close}, $\Gamma_{\text{far}}(q_{\text{close}})$, as new candidates to update $\Gamma_{\text{far}}(p)$. In addition, the procedure pushes towards q_{close} the far neighbors of p, as nodes far from p are likely to lay far from q_{close} as well. The receipt of the corresponding far message is handled by the code shown in Algorithm 5.
- farCandidatesCloseToFar(p) (Algorithm 4) implements the *close-to-far* strategy presented above: this time, p picks one of its current far neighbors q_{far}, and returns the close neighbors of q_{far}, $\Gamma_{\text{close}}(q_{\text{far}})$ in order to improve $\Gamma_{\text{far}}(p)$. The procedure also pushes towards q_{far} the close neighborhood of node p, $\Gamma_{\text{close}}(p)$, as those are likely to lay far from q_{far}. The push message, of type far, is handled as above.
- farCandidatesMixed(p) (Algorithm 6) combines the two above strategies in one single heuristics. As in Algorithm 2, we use the **with probability** construct to switch between the *far-from-close* and *close-to-far* strategies with probability β, thus insuring that only one push-pull exchange occurs every time farCandidatesMixed(p) is invoked. The parameter β further controls how much each strategy is used, and allows farCandidatesMixed(p) to generalize the previous two procedures: the extreme case $\beta = 0$ corresponds to the *far-from-close* strategy, while $\beta = 1$ implements a *close-to-far* approach.

Considered all-together, Algorithms 2 to 6 capture a family of decentralized k-furthest-neighbor (KFN) graph construction protocols, controlled by two stochastic parameters, α and β. Parameter α controls the distribution of efforts between the intermediate KNN view and the final KFN view, while β arbitrates between the *far-from-close* and *close-to-far* strategies.

4 Evaluation

We evaluate our framework using the simulator PeerSim [23], and compare its behavior against a basic greedy epidemic protocol (Algorithm 1) that uses a negative similarity metric (Equation 1). We term this baseline solution *Far From Far*. We are essentially interested in two aspects of our solution: (i) *its convergence*, i.e. how fast our framework is able to converge to a good KFN graph, and (ii) *its scalability*, i.e. how does this convergence speed evolve with growing network sizes. The code used for our experiments can be found on-line at https://gitlab.inria.fr/ASAP/HyFN.

4.1 Experimental set-up and metrics

Unless stated otherwise our default set-up involves 3200 nodes regularly positioned on a $[0, 1)$ ring. By default, we use views of $k = 14$ nodes, and fetch $r = 3$ random nodes in each round. We set the parameters of HyFN to $\alpha = \beta = 0.5$.

These values mean that on average nodes spend the same number of rounds constructing their KNN and KFN views (α at line 3 of Algorithm 2), and that the construction of the KFN view uses the heuristics *far-from-close* and *close-from-far* in equal measure (β at line 2 of Algorithm 6). We assume a random peer sampling service (RPS) [18] is available, which we use to initialize all views with random nodes before the protocol starts, and to provide r random nodes in each round.

To measure the convergence of the approximate KFN graph constructed by HyFN we use the following four metrics:

- **Number of missing links**: We count for each node how many of its k furthest neighbors are missing from its KFN view. The count of all these *missing links* over the network yields our first metric.
- **Number of converged nodes**: As a second measure of convergence, we consider that a node is converged when at least 80% of its k furthest neighbors (taking into account ties) are contained in its KFN view. As a measure of the network's convergence, we count in each round how many nodes are converged.
- **Average KFN distance**: For each node, we compute the average distance between this node and the nodes in its KFN view. This metric should tend toward 0.5 in a ring of perimeter 1 (our default topology). Note that even a perfectly converged network won't actually reach 0.5 though, with the exact value depending on the density of the network.
- **Convergence time** Finally, we consider that the whole network is converged when at least 80% of all nodes are converged, according to the above criterion. We count the number of rounds until this convergence condition is fulfilled.

We do not report the communication overhead of either HyFN or our baseline: the protocols are all designed to initiate one single push-pull exchange in each round, and therefore present the same communication costs.

In the following we first evaluate HyFN on our default scenario (3200 nodes on a regular ring, $k = 14$, $r = 3$, $\alpha = \beta = 0.5$) and compare it against our baseline. We then analyze the impact of the mixing parameters α and β. Finally, we study the scalability of HyFN up to networks of 12800 nodes, both on a ring and grid topology. All reported values are averages computed over 25 experimental runs.

4.2 Results

Figure 3 shows the convergence of HyFN in our default scenario (3200 nodes on a regular ring), according to three convergence metrics: the percentage of converged nodes (Figure 3a), the number of missing links (Figure 3b), and the average KFN similarity (normalized to 1, Figure 3c). The behavior of three variants of HyFN are shown, which correspond to the three heuristics presented in Algorithms 3 (*Far-from-Close*), 4 (*Close-to-Far*), and 6 (*Hybrid*), discussed in Section 3.2.

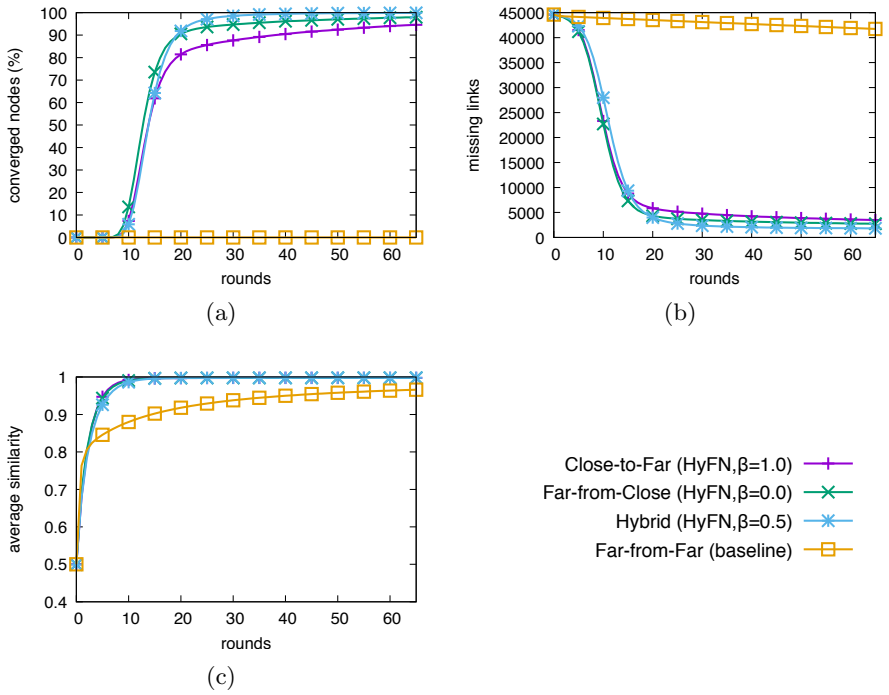
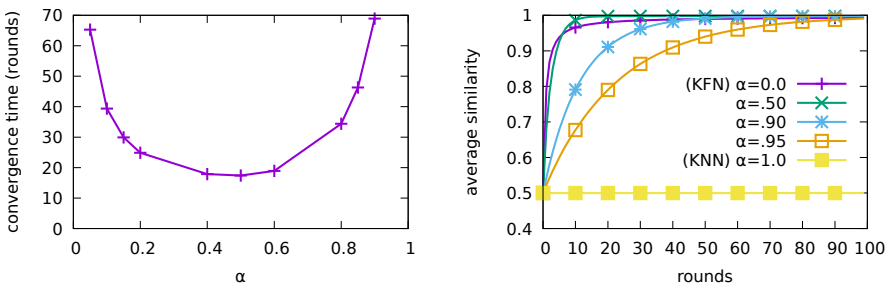


Fig. 3. Converged nodes, missing links, and average similarity of the baseline (*Far-from-Far*) and of three versions of HyFN (corresponding to $\beta = 1$ for *Close-to-Far*, $\beta = 0$ for *Far-from-Close* and $\beta = 0.5$ for *Hybrid*) on a 3200-node regular ring.

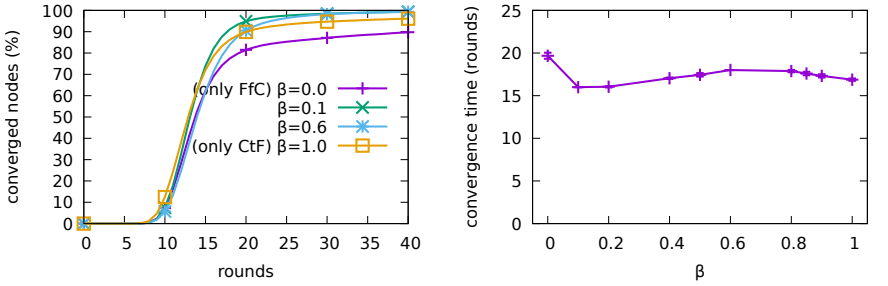
Comparison to the Far-from-Far baseline. From the three convergence metrics, it appears that the three versions of HyFN clearly outperform the baseline. More precisely, all HyFN variants have reached 80% of converged nodes after at most 20 rounds whereas the baseline is unable to converge even after 65 rounds (Figure 3a). Interestingly, the hybrid variant has the best performances in terms of overall convergence. From the average similarity metric (Figure 3c), the baseline has the worst performances, even if it gets decent results in a reasonable time. In fact, it doesn't get the farthest neighbors, but still it gets far neighbors. Moreover, the metric of missing links (Figure 3b) shows clearly that the baseline does not work: it just converges linearly only due to the couple of random neighbors that are fetched at each turn. Finally, among all HyFN variants, the *Hybrid* approach seems to converge most closely to the theoretically ideal network at the price of being a slightly slower than *Close-to-Far*.

Influence of the parameters α and β. Our key aim is to evaluate the effective impact of the stochastic parameters α and β on the KFN graph and to set them accordingly. Figure 4 outlines the impact of the α parameter, and



(a) (b)

Fig. 4. Impact of the α stochastic parameter on a 3200-node regular ring.



(a) (b)

Fig. 5. Impact of the β stochastic parameter on a 3200-node regular ring.

shows that $\alpha = 0.5$ is close to the optimal. This value provides: (i) the best convergence time (Figure 4a), and (ii) the best tradeoff between the convergence speed and the quality of the neighborhood (Figure 4b). Concerning the impact of fine tuning β (Figure 5), having β close to 0.2 gives the best network convergence, and convergence speed. Note that, we are not able to reach 100% of converged nodes when we choose a β value of either 0 or 1. As a result having a non hybrid heuristic is not the most suitable choice, although the results of these kind of heuristics is still better than the baseline. Furthermore, as soon as we use the hybrid strategy, the value of $0 < \beta < 1$ has a little impact on the convergence time.

Consequently, it appears that fine tuning α is predominant compared to β. In other terms, once we have set α to its best value (i.e 0.5), the value of β has a little impact as long as $0 < \beta < 1$, so as long as we are actually using an hybrid approach.

Scalability. We have investigated the applicability of the hybrid heuristic on both a ring and grid logical networks of varying sizes from 100 to 12800 nodes

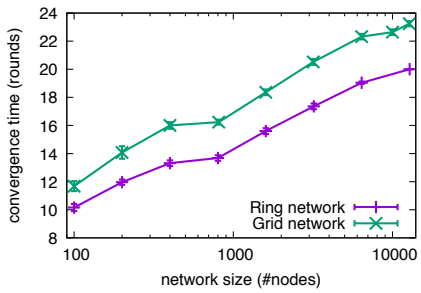


Fig. 6. Behavior of HyFN with the hybrid heuristic for networks from s = 100 nodes to s = 12800 nodes, for a variety of network topologies (Ring and Grid in the above figure).

(Figure 6). When the size s of the network is modified, we scaled k and r accordingly such that $k = 1.2 * \log_2(s)$ and $r = 0.3 * \log_2(s)$, both rounded to the closest integer — in order to give back $k = 14$ and $r = 3$ for $s = 3200$. As a result, it appears that HyFN converges as expected in logarithmic time relative to the network total size, demonstrating thus that our approach scales well.

5 Related work

To the best of our knowledge, HyFN is the first decentralized protocol specifically designed to compute a distributed k-furthest-neighbor (KFN) graph.

In terms of related mechanisms, a distributed KFN graph is a form of peer-to-peer network overlay. Peer-to-peer overlays have been widely applied in the past to implement distributed services, ranging from distributed storage [25, 26, 28], and streaming [12, 22], through to pub/sub [2, 24] and environmental sensing [15]. Among peer-to-peer overlays, k-nearest neighbor (KNN) overlays [17, 30, 4] come closest to HyFN, although they converge poorly when applied to the KFN graph construction problem, as our evaluation shows. KNN overlays have been extensively studied in the past, as they provide decentralized self-organization properties which have been exploited to implement a large number of resilient and scalable services, from recommendation systems [4, 14, 30], to collaborative caching [11] and generic topology construction [5, 17].

Epidemic topology construction protocols such as the ones presented in this work are typically highly scalable and efficient due to their inherent concurrency (each node executes the protocol in parallel) and locality (nodes only perform a few interactions per round). These two properties (concurrency and locality) render these algorithms also attractive for high-end parallel machines, and have given rise to several highly effective parallel KNN graph construction algorithms [7, 8, 10].

VM placement (the main technique for data centre optimisation) aims at assigning VMs to PMs in data centres — so that some cost function(s) is minimised [3, 27], such as, electricity cost, resource (e.g. CPU or memory) wastage,

maintenance cost. The problem is often described as an instance of the general bin packing problem, and most techniques in the literature pack as many VMs as possible on PMs. However, in practice, piling up VMs may not be such a good idea as all resources cannot be perfectly isolated. This lack of isolation generates contentions between VMs hosted in the same PM; for instance, pressure on cache or I/O by one VM will have an impact on the other VMs sharing this PM. Most studies in the literature use time series analysis to compare two VMs' workloads. For instance, Halder et al. [16] propose an interference aware first fit decreasing using a large correlation matrix – keeping track of the VMs' time series and the composition of those time series in each PM. Verma et al. [29] simplify the time series using a concept of *envelop*, recording only the peaks of utilisation and not the full time series. They then cluster similar workloads and make sure they do not end up in the same PMs. Li et al. [21] propose a two phase clustering that addresses the scalability issues that previous approaches suffer from. They also propose a placement algorithm that minimises the number of required PMs and the number of interferences. Their solution[3] would certainly benefit from the concept of KNN and the algorithms proposed in the current paper – we are working on an adaptation to an industry setting, with large and hosting departments running complex workloads.

6 Conclusion

In this paper, we propose HyFN, a novel and generic decentralized protocol to compute k-furthest-neighbor (KFN) graphs. HyFN exploits an intermediate k-nearest-neighbor (KNN) graph, which is constructed in parallel, to progressively converge towards an optimal solution. We have in particular proposed three heuristics to exploit this KNN graph. Our evaluation shows that our proposal clearly outperforms a naive greedy implementation based on existing KNN epidemic protocols.

Beyond its application to decentralized and pair-to-pair systems, we believe our KFN construction framework holds a strong potential for the computation of KFN graphs on highly parallel machines. Its inherent properties of locality and high concurrency are likely to make it a worthwhile approach in cases in which a KFN graph is required, including resource allocation problems such as those encountered in VM allocation services.

Acknowledgement

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 to Lero - the Irish Software Research Centre (www.lero.ie), and by the ANR (Agence Nationale de la Recherche) Project PAMELA n. ANR-16-CE23-0016.

[3] Note that one of the co-authors of the present paper was senior author of [21].

References

1. X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Gossiping personalized queries. In *EDBT'2010*.
2. R. Baldoni, R. Beraldi, V. Quéma, L. Querzoni, and S. T. Piergiovanni. TERA: topic-based event routing for peer-to-peer architectures. In *DEBS 2007*, pages 2–13. ACM, 2007.
3. A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *FGCS*, pages 755–768, 2012.
4. M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Middleware'2010*.
5. S. Bouget, H. Kervadec, A.-M. Kermarrec, and F. Taïani. Polystyrene: the decentralized data shape that never dies. In *ICDCS'14*, pages 288–297, June 2014.
6. A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec. WhatsUp Decentralized Instant News Recommender. In *IPDPS*, 2013.
7. A. Boutet, D. Frey, R. Guerraoui, A.-M. Kermarrec, and R. Patra. HyRec: Leveraging Browsers for Scalable Recommenders. In *Middleware*, pages 85–96, 2014.
8. A. Boutet, A.-M. Kermarrec, N. Mittal, and F. Taïani. Being prepared in a sparse world: the case of KNN graph construction. In *ICDE'16*, pages 172–181, 2016.
9. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *PODC'87*.
10. W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, pages 577–586, 2011.
11. D. Frey, M. Goessens, and A.-M. Kermarrec. Behave: Behavioral cache for web content. In *IFIP DAIS'14*, pages 89–103, 2014.
12. D. Frey, R. Guerraoui, A.-M. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma. Heterogeneous gossip. In *Middleware'09*, pages 42–61, 2009.
13. D. Frey, A. Jégou, and A.-M. Kermarrec. Social Market: Combining Explicit and Implicit Social Networks. In *SSS'11*, Grenoble, France, Oct. 2011. LNCS.
14. D. Frey, A.-M. Kermarrec, C. Maddock, A. Mauthe, P.-L. Roman, and F. Taïani. Similitude: Decentralised adaptation in large-scale P2P recommenders. In *IFIP DAIS'15*, pages 51–65, Grenoble, France, 2-4 June 2015.
15. P. Grace, D. Hughes, B. Porter, G. S. Blair, G. Coulson, and F. Taïani. Experiences with open overlays: a middleware approach to network heterogeneity. In *Eurosys'08*, pages 123–136, Glasgow, Scotland UK, 31 Mars–4 April 2008. ACM.
16. K. Halder, U. Bellur, and P. Kulkarni. Risk aware provisioning and resource aggregation based consolidation of virtual machines. In *CLOUD*, pages 598–605, 2012.
17. M. Jelasity, A. Montresor, and O. Babaoglu. T-man: Gossip-based fast overlay topology construction. *Computer networks*, 53(13):2321–2339, 2009.
18. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen. Gossip-based peer sampling. *ACM ToCS*, 25(3):8, 2007.
19. A.-M. Kermarrec and F. Taïani. Diverging towards the common good: heterogeneous self-organisation in decentralised recommenders. In *SNS'2012*.
20. J. Koomey and J. Taylor. New data supports finding that nearly a third of capital in enterprise data centers is wasted, 2015.
21. X. Li, A. Ventresque, J. O. Iglesias, and J. Murphy. Scalable correlation-aware virtual machine consolidation using two-phase clustering. In *HPCS*, pages 237–245, 2015.

22. J. Liu and M. Zhou. Tree-assisted gossiping for overlay video distribution. *Journal of Multimedia Tools and Applications*, 29(3):211–232, 2006.
23. A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *P2P'09*.
24. J. A. Patel, E. Riviere, I. Gupta, and A. Kermarrec. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Computer Networks*, 53(13):2304–2320, 2009.
25. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01*, pages 161–172, New York, NY, USA, 2001. ACM.
26. A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01*, pages 329–350, London, UK, UK, 2001. Springer-Verlag.
27. T. Saber, A. Ventresque, I. Brandic, J. Thorburn, and L. Murphy. Towards a multi-objective vm reassignment for large decentralised data centres. In *UCC*, 2015.
28. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.
29. A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *USENIX ATC*, pages 28–28, 2009.
30. S. Voulgaris and M. v. Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par'05*.
31. F. Xu, F. Liu, H. Jin, and A. V. Vasilakos. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. *Proceedings of the IEEE*, 102(1):11–31, 2014.